UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Centre de la Imatge i la Tecnologia Multimèdia

Creación de un juego pixel art 2D *usando modelos 3D*

Trabajo final de grado
Grado en Diseño y Desarrollo de Videojuegos

Apellidos: Cabreira Sánchez
Nombre: Carlos
Plan: 2014

Director*:* Ripoll Tarré, Marc

Índice

Palabras Clave

Pixel art, pipeline 3D, Blender, Unity, game development, 3D model, 2D art, hack and slash

Índice de figuras

Índice de tablas

Resumen

Crear arte en dos dimensiones para videojuegos es lento. Parte del problema está en que el desarrollo de un juego es un proceso altamente iterativo, donde se descartan y renuevan ideas cada poco tiempo. Teniendo en cuenta que las animaciones se generan, tradicionalmente, dibujando cada fotograma a mano, rehacerlas supone una enorme inversión de trabajo.
Muchos estudios optan por la técnica pixel-art no solo por cuestiones estéticas, sino porque, al trabajar en una resolución muy baja (16x16 píxeles, 32x32, 256x256...), producir las animaciones no requiere tanto tiempo.Aun así, el dibujar a mano cada fotograma sigue siendo una carga de trabajo enorme. Esto genera un cuello de botella importante, ralentizando el desarrollo, dificultando la iteración y, como consecuencia, disminuyendo la calidad del producto final.
El objetivo de este trabajo era demostrar cómo acelera y facilita el proceso de crear arte pixel-art en 2D el uso de modelos 3D para renderizar los sprites (lo que se denomina *prerrenderizado a baja resolución*), y comprobar el impacto que esta técnica tiene en el desarrollo global.
Para ello, se ha llevado a cabo el desarrollo de un pequeño videojuego, siguiendo la metodología de *preproducción - producción - postproducción*, y usando el motor de juegos Unity. El juego ha sido desarrollado por una sola persona (a excepción de la música), y tiene una duración media de 30 minutos.

El ahorro de tiempo que el prerrenderizado ha traído a este desarrollo ha sido mayúsculo: tareas que, de manera tradicional, hubieran ocupado días, se redujeron a una cuestión de minutos. Aunque ciertamente existen ciertas áreas de mejora, el resultado ha sido muy positivo.

En este documento se profundizará en diversos aspectos técnicos y de diseño, tanto del juego en sí como de la técnica del prerrenderizado en baja resolución.

El ejecutable del juego se puede descargar desde Itch.io, de manera gratuita : https://carlos-cabreira.itch.io/demon-overdrive

Glosario

Pixel-art: Estilo artístico 2D donde se usan imágenes de muy baja resolución (normalmente, entre 16x16 y 256x256 píxeles), con el fin de imitar el estilo de los juegos de los años 80/90.

Pipeline/workflow: Metodología de trabajo, pasos establecidos para generar contenido

Hack and Slash: Género de videojuego de acción donde, normalmente usando armas de filo, se combate contra numerosos enemigos.

Game loop: Posibilidad de terminar la partida dentro del juego y comenzar de nuevo sin necesidad de salir de la aplicación (y con opciones de salir al menú principal, de cerrar el juego, etc).

Bug: fallo del software, algo que no funciona como debería. Pueden ser de baja gravedad (una textura que desaparece) o de alta gravedad (la aplicación se cierra involuntariamente).

Placeholder: versión temporal de algo, usada como indicación de que, en el producto final, se tendrá que actualizar a la versión definitiva (por ejemplo, un cubo para indicar que, en un futuro, un objeto tendrá que figurar en su lugar).

Vertical slice: pequeña demostración jugable que contiene todas las mecánicas del juego completo. Se usa para hacerse una idea de cómo será el producto el día de su lanzamiento. Es importante que el acabado artístico sea lo más parecido al final, y que todos los elementos que vayan a estar en el juego completo tengan su pequeña representación aquí.

Indie: abreviatura de “independiente”. Se refiere a juegos (normalmente de recursos limitados y equipo de pequeño tamaño) que han sido desarrollados sin la intervención de grandes compañías.

1. Introdución

1.1 Motivación

A pesar de mi formación como programador, siempre he tenido un cierto interés en todo lo que rodea al arte en 3D: modelaje, texturizado, programación gráfica, etc.
El germen de este proyecto fue un artículo públicado en Gamasutra por Thomas Vasseur, artista en el estudio Motion Twin. En dicho artículo, Vasseur explicaba como habían usado modelos 3D para crear el arte 2D de Dead Cells, su último y más exitoso lanzamiento, debido a la falta de personal.

Dicha técnica (conocida como prerrenderizado), en palabras de Thomas Vasseur, permitia la rápida creación de contenido, y su aún más rápida iteración, ahorrando cantidades masivas de tiempo.

Movido por la curiosidad, y por el hecho de que llevaba tiempo queriendo probar mis habilidades de programación y diseño, decidí empezar este proyecto, aplicando la misma metodología que Thomas Vasseur había empleado en Dead Cells.

1.2 Formulación del problema

La creación de animaciones en 2D es un proceso pesado donde, normalmente, cada fotograma se ha de hacer de manera individual y a mano. Dentro del desarrollo de un videojuego (que requiere de mucho ensayo y error), esto puede ocasionar cuellos de botella importantes. Incluso los mas pequeños cambios (modificar el color de un personaje, sustituir un arma por otra, corregir la velocidad de una animación) requieren una tremenda inversión de tiempo.

Muchos estudios (especialmente, los de pequeño tamaño), optan por utilizar la conocida técnica *pixel-art*, no solo para imitar la estética de juegos de antes de los 2000, sino porque, al trabajar en resoluciones muy inferiores a las habituales (y forzar la pérdida de detalle), se acelera el proceso de manera significativa.
Aún así, este sigue siendo un proceso relativamente costoso, donde un solo artista puede invertir horas (e incluso días) en una sola animación.
En el siguiente video se puede observar (de manera acelerada) el proceso de crear una animación pixel-art de manera tradicional.

1.3 Objectivos generales del TFG

- Estudiar la técnica artística de *Dead Cells,* aplicarla en la creación de una v*ertical slice* (un nivel + un enemigo final) de un juego, y observar el impacto que este *workflow* tiene en su desarrollo, en términos de calidad y ahorro de tiempo.
- Experimentar y mejorar dicha técnica: analizar pros y contras, aplicarla en distintos ámbitos (animación, escenarios, menús, etc) y extraer conclusiones.
- Alcanzar una calidad artística aceptable para un juego comercial.
- Aplicar los conocimientos de programación, arte y diseño adquiridos en el grado.
- Publicar el juego en Itch.io (página web de publicación de juegos independientes) de manera gratuita.

1.4 Objetivos específicos del TFG – Acerca del juego

El juego a desarrollar, nombrado como *Demon Overdrive*, consiste en un título de acción *hack and slash* 2D, en vista cenital, con estilo artístico pixel-art. Inspirado en títulos como *NieR: Autómata* o *Bayonetta*, es un juego centrado en el combate rápido y agresivo, donde se le proporciona al jugador diversas habilidades y movimientos con los que enfrentarse (de manera normalmente espectacular y llamativa) a un gran número de enemigos.

Los pilares fundamentales del juego son:

- ***Centrado en el combate cuerpo a cuerpo:*** sin diálogos, items ni exploración.
- ***Siempre en movimiento***: todas las habilidades del jugador lo exhortan a desplazarse constantemente por el escenario. La posición de los enemigos en pantalla es crucial.
- ***Enemigos modulares***: cada unidad enemiga tiene un comportamiento sencillo y característico, y generan nuevas dinámicas si se combinan en un mismo nivel.
- ***Satisfacción immediata:*** el mero hecho de ponerse a los mandos y xontrolar al personaje principal debe resultar placentero al jugador.

1.5 Alcance del proyecto

- Un nivel jugable, con principio y fin, de *al menos* media hora de juego.
- Cinco tipos distintos de enemigos.
- Un jefe final que sirva de cierre.
- Interfaz de usuario básica: menú de inicio, de pausa/muerte y de créditos. Opciones más complejas (como configuración de controles, de gráficos y similares) no estarán contempladas, para ajustarse al tiempo disponible
- Pequeños momentos narrativos no interactivos (uno al comienzo y otro al final).

1.5.1 Posibles obstáculos

El principal riesgo en este proyecto (como viene siendo natural en el desarrollo de juegos) es no disponer de suficiente tiempo para llevar a cabo las tareas planeadas. Esta situación se puede contrarrestar o paliar con un calendario estricto y un plan de contingencias realista. Ambos se pueden encontrar más abajo en este documento.

A pesar de ello, y dado que la finalidad de este TFG es experimentar con una técnica que acelera un proceso, el hecho de disponer de márgenes temporales ajustados resulta ser un buen contexto de estudio.

Otro potencial obstáculo es que el *workflow 3D* no cumpla las expectativas de efectividad, pudiéndose cumplir cualquiera de estos escenarios:

- El ahorro de tiempo no es significativo: la diferencia entre realizar una tarea de manera tradicional y con el nuevo *workflow* es menor a media hora (aproximadamente).
- La estética artística no resulta convincente para el usuario (ya sea por perdida de detalle, problemas de fluidez o aparición de errores gráficos recurrentes). Al tratarse de una apreciación subjetiva, este hecho deberá corroborarse durante los testeos (detallados más abajo en este documento).
- El *workflow 3D* tiene demasiadas limitaciones (errores de perspectiva, falta de detalle o control, etc) para ser aplicado en proyectos de mayor tamaño.

1.5.2 Público objetivo

Este proyecto se podría dividir en dos vertientes, cada una con su público objetivo particular:

- **El juego**: el ejecutable final, perfectamente jugable, que será publicado en ***Itch.io*** (plataforma online de títulos independientes y/o experimentales) de manera gratuita al finalizarse. Su público objetivo son jugadores que busquen títulos experimentales, preferentemente de acción.
- **La investigación sobre el Workflow 3D**: todas las conclusiones obtenidas tras desarrollar un juego con esta técnica, los detalles de su implementación, y posibles mejoras. Su objetivo son otros desarrolladores que planeen crear juegos pixel-art y que dispongan de pocos recursos.

2. Estado del arte

La técnica de prerrenderizar gráficos 3D existe desde *Donkey Kong Countr*y (SNES, 1994), donde se usaba para conseguir gráficos detallados sin exceder las capacidades tecnológicas de la consola.

Captura de pantalla de Donkey Kong Country

A partir de ahí, no han sido demasiados los juegos que han usado esta técnica (entre ellos, tenemos los primeros títulos de la saga Resident Evil, que la usaba en sus escenarios). El avance de la tecnología ha convertido en obsoleta esta técnica, ya que las plataformas de hoy en día son capaces de mover sin problemas personajes poligonales y escenarios ultradetallados.
Sin embargo, *Dead Cells* propone una aproximación diferente: en lugar de usar la prerrenderización para ganar detalle, la usa para, deliberadamente, perder detalle, y falsear una estética *retro* que requiere de menos tiempo de desarrollo.

A mayores de este título, no se ha encontrado ningún otro juego popular que aplique la misma técnica gráfica, por lo que se puede certificar que Dead Cells es un juego pionero en este aspecto (incluso los títulos que Thomas Vasseur menciona como referencia, *Blazblue* y *Guilty Gear Xrd,* usan el prerrenderizado de manera tradicional).

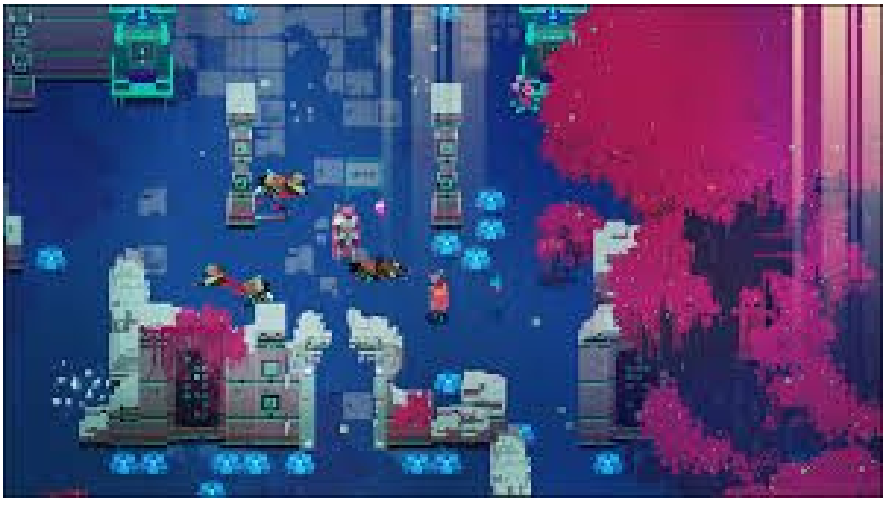
2.1 Estudio de Mercado

El nicho de mercado al que Demon Overdrive quiere pertenecer, cumple con la siguientes características:

- Indie
- Acción rápida y brutal, centrada en el combate cuerpo a cuerpo
- Pixel art
- Vista cenital
- Estilo artístico llamativo

Los principales competidores (Indies de acción en 2D pixel-art), son (junto a su desarrolladora, su año de lanzamiento y enlace a su página web):

Hyperlight Drifter (Heart Machine, 2016)



Hotline Miami (Dennaton Games, 2012)

Moonlighter (DigitalSun, 2018)

Dead Cells (Motion Twin, 2017)

Sin embargo, ninguno de los aquí presentados se solapa completamente con *Demon Overdrive*.

Hyper Light Drifter se centra más en la exploración, y su combate es lento; H*otline Miami* es un juego de disparos de estética feísta, algo completamente opuesto a *Demon Overdrive*; *Moonlighter* posee elementos de gestión y estrategia, y su combate, al igual que *Hyper Light Drifter*, es sencillo y de ritmo moderado.
El caso más similar a *Demon Overdrive* es *Dead Cells*.

Tradicionalmente, el ritmo de combate de los juegos *pixel*-art se ve lastrado por las propias limitaciones de la técnica tradicional (animaciones con pocos fotogramas, movimiento aparatoso, etc).
Dead Cells se libra de estas limitaciones y consigue un ritmo increíblemente veloz gracias al workflow en 3D. Sin embargo, su gameplay se enfoca más a las plataformas, a la repetición y al diseño procedural de niveles, elementos que no comparte *Demon Overdrive*.
El objetivo de *Demon Overdrive* es, por lo tanto, conseguir el frenético ritmo y brutalidad de *Dead Cells* en un género completamente diferente, creándose un segmento de mercado propio.

3. Metodología

Para el desarrollo de Demon Overdrive, se ha optado por una metodología de Preproducción, producción y post-producción, ya que es clara, sencilla y eficaz; aparte de ser habitual en el desarrollo de software. Adicionalmente, previa a la fase de Preproducción (para facilitar el desglose de tareas), se incluye Concept discovery.

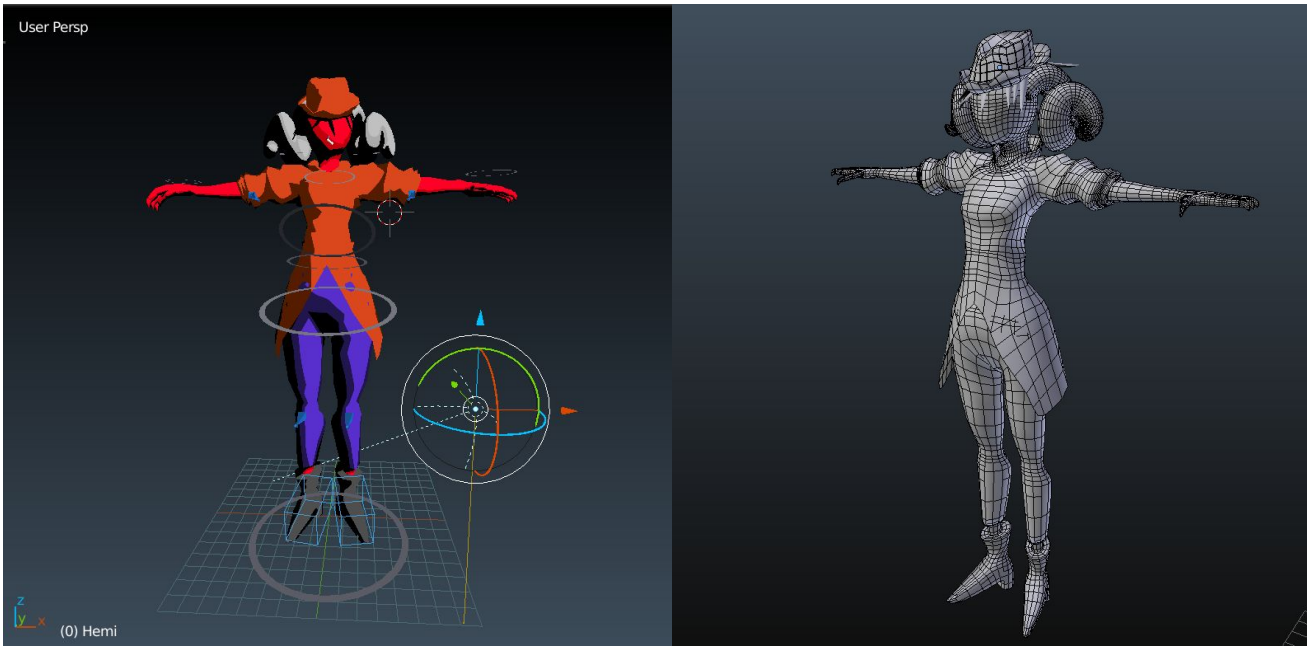
3.1 Desglose

- ***Concept discovery***: se plantean diversas ideas para el juego (género, pilares base), y se descartan y perfilan hasta acotar las que serán incluidas en un prototipo.
- ***Preproducción (versiones 0.0.x)***: se crea el prototipo, y se hacen las primeras pruebas con la pipeline 3D. Se analizan ventajas y desventajas, y se establece una metodología de cara a producción. También se hacen pruebas artísticas. Se programan las mecánicas base, que se testean. Posteriormente, se establece cuales son las que se introducirán en producción y cuales se descartarán. El objetivo es entrar en producción con un pequeño nivel jugable, con un solo tipo de enemigo.
- ***Producción (versiones 0.1.0 hasta 0.7.0)***: se generan los enemigos, niveles y demás elementos jugables. Las mecánicas base deberán estar fijadas previamente. Al finalizar la fase, todo el contenido del juego debe estar incluido, aunque se aceptan bugs y placeholders. La versión que salga de esta fase se denominará Alpha.
- ***Post-producción (versiones 0.7.0 hasta 1.0.0)***: no se generará más contenido en esta fase. Se arreglan bugs, se incluye el arte definitivo y se llevan a cabo labores de pulido. La versión final 1.0 se denominará Gold, y se considerará el desarrollo finalizado.

3.2 Creación de personajes con el *workflow 3D*

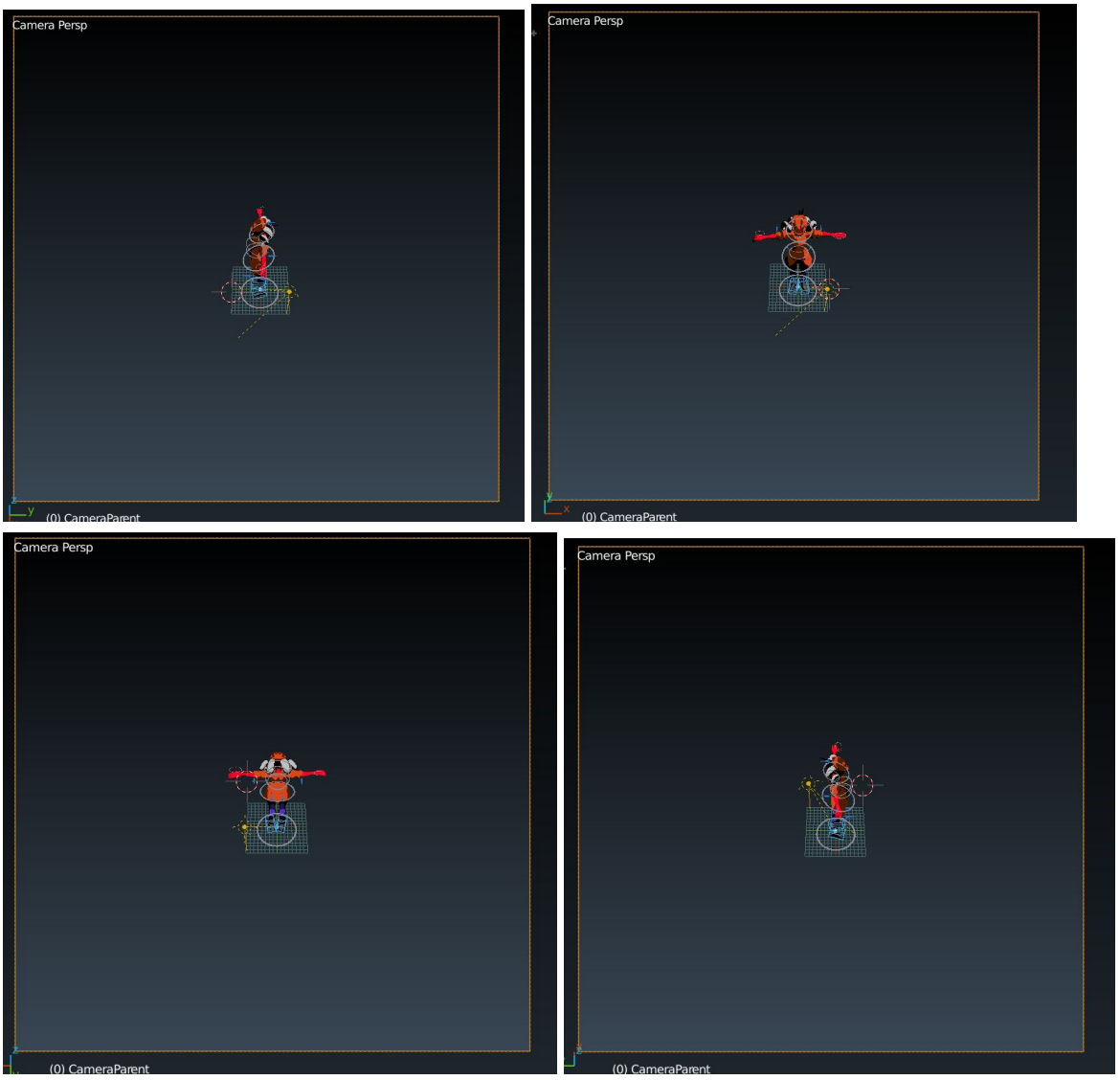
Todo este proyecto orbita en torno al ya mencionado *Workflow 3D* o *prerrenderizado*, pero, ¿en qué consiste este método?
Estos son los pasos seguidos para crear a los personajes de *Demon Overdrive:*

- Se crea el modelo 3D del personaje. La cantidad de detalle debe ser moderada, ya que se más adelante puede perderse. En el caso de *Demon Overdrive*, la herramienta usada es **Blender.**
- Se crea el esqueleto que servirá para animar al personaje. Este proceso no se diferencia de crear un esqueleto para animación 3D tradicional.
- Se colorea el personaje. No es necesario usar texturas (ya que no se apreciarán bien en el resultado final). Es recomendable evitar sombreados realistas e inclinarse por colores planos y sombras duras.



Modelo primitivo de la protagonista de *Demon Overdrive (1712 vertices)*

- Se establece desde qué ángulo se va a procesar la animación. En el caso de *Demon Overdrive*, cada animación se procesa desde cuatro ángulos diferentes:

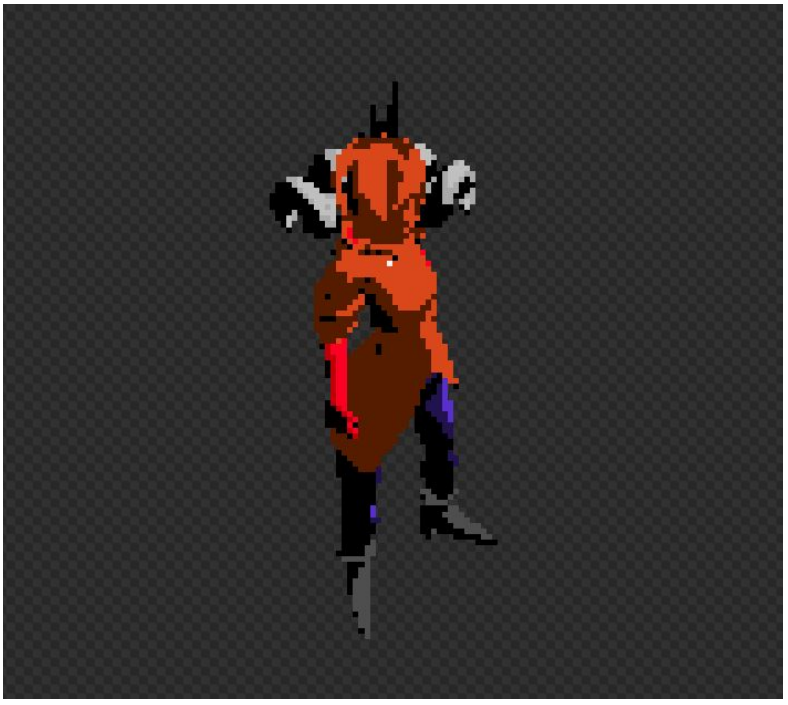


Los cuatro ángulos de cámara desde los que se procesan los modelos en *Demon Overdrive*

Para acelerar el proceso, se ha creado un pequeño código en lenguaje Python que, en tiempo de procesado, se encarga de rotar la cámara entorno al personaje:

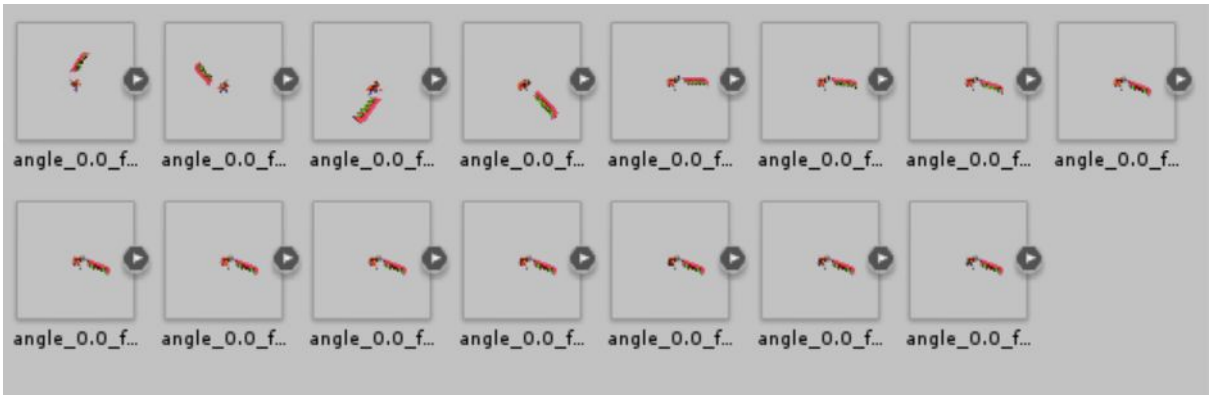
```python
import bpy
from math import radians
from os.path import join

S = bpy.context.scene

anim_name = '\ChargeAttack'

originalRenderFolder = bpy.context.scene.render.filepath


camParent = bpy.data.objects['CameraParent']

animLen   = 15 # frames
numAngles = 4
rotAngle  = 360 / numAngles
camParent.rotation_euler.z = radians( 0 )
for i in range(numAngles):

    # Set camera angle via parent
    angle = i * rotAngle
    camParent.rotation_euler.z = radians( angle )

    # Set the proper renderfolder
    renderFolder = originalRenderFolder
    renderFolder += anim_name

    if(i == 0):
        renderFolder+='\Front'
    if(i == 1):
        renderFolder+='\Left'
    if(i == 2):
        renderFolder+='\Back'
    if(i == 3):
        renderFolder+='\Right'

    # Render animation
    for f in range(1,animLen + 1):
        S.frame_set( f ) # Set frame

        frmNum   = str( f ).zfill(3) # Formats 5 --> 005
        fileName = "angle_{a}_frm_{f}".format( a = angle, f = frmNum )
        fileName += S.render.file_extension
        bpy.context.scene.render.filepath = join( renderFolder, fileName )

        bpy.ops.render.render(write_still = True)

bpy.context.scene.render.filepath = originalRenderFolder
camParent.rotation_euler.z = radians( 0 )
```

- El personaje se anima a una velocidad de 30 fotogramas por segundo, tal y como se animaría un modelo 3D tradicional.
- Se procesa la animación entera a baja resolución (en caso de *Demon Overdrive*, se usan resoluciones de 256x256 o 512x521 píxeles, dependiendo del personaje)



Resultado de procesar el modelo anterior a 521x521 píxeles

Finalmente, estas imágenes se importan en el motor de juegos **Unity** como animaciones 2D tradicionales, y se manejan como tal a partir de ahora.



Fotogramas de una animación, importados en Unity

Este proceso fue creado y refinado durante la fase de pre-producción (incluyendo el modelo y el código mostrados).

3.3 Creación de escenarios

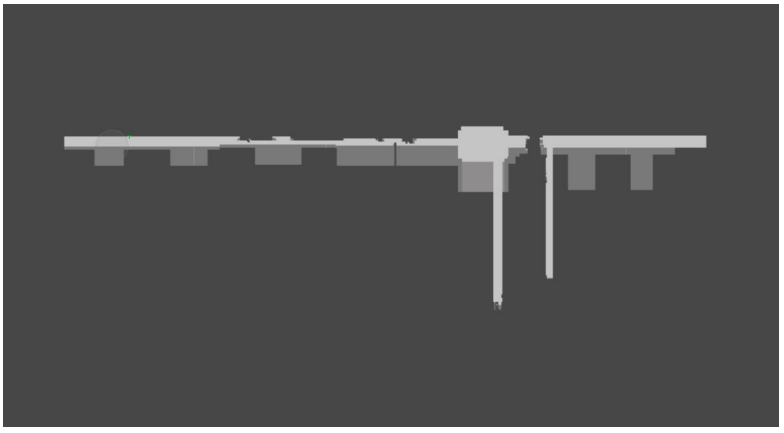
El diseño y la creación de los escenarios (también 2D) sigue el siguiente patrón:

- Se genera una tabla donde se deciden y analizan los diferentes puntos de desafío de cada pantalla. Se decide qué tipo de enemigos se añadirán, cuántos, que tipo de comportamiento se busca que tenga el jugador, etc. La S representa la introducción de un nuevo tipo de reto (en este caso, la primera aparición de un nuevo enemigo). Las X, que ese desafío es una evolución por replicación (se añaden más enemigos ya conocidos), y las E, que en una evolución por variación (mismas mecánicas, pero nuevas variaciones, que pueden venir de, por ejemplo, cambios en la colocación en el escenario)

	A	B	C	D	E	F	G	H
1	***Screen 3***							
2								
3								
4	**Challenge number**	1	2	3	4	5	6	7
5								
6								
7	Guard	X			X			X
8	Mage						X	X
9	Eye		S	X	X	E	X	E
10	Phantom							

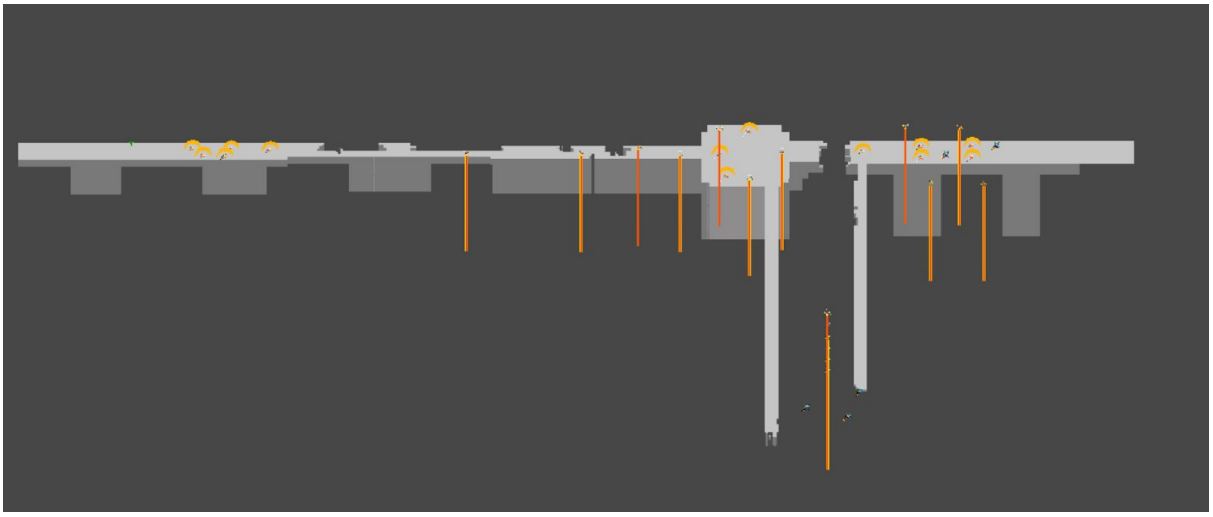
Tabla de enemigos del tercer nivel

- Se procede a generar un mapa básico en Unity, usando la herramienta Tileset (que permite generar rápidamente mapas jugables)



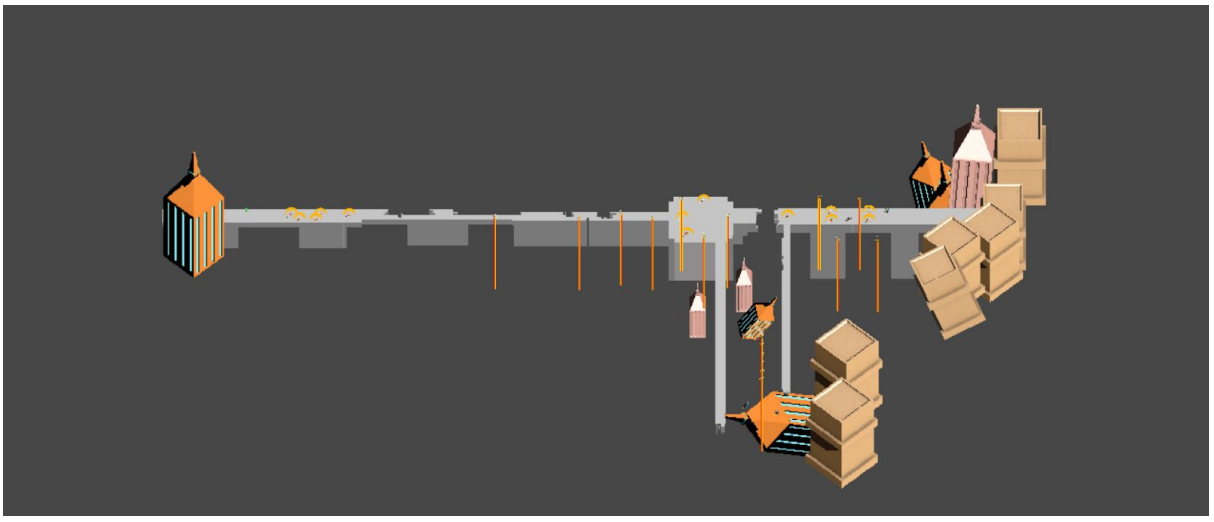
Esqueleto del tercer nivel

- Se incorporan enemigos al mapa, siguiendo el patrón creado en las tablas



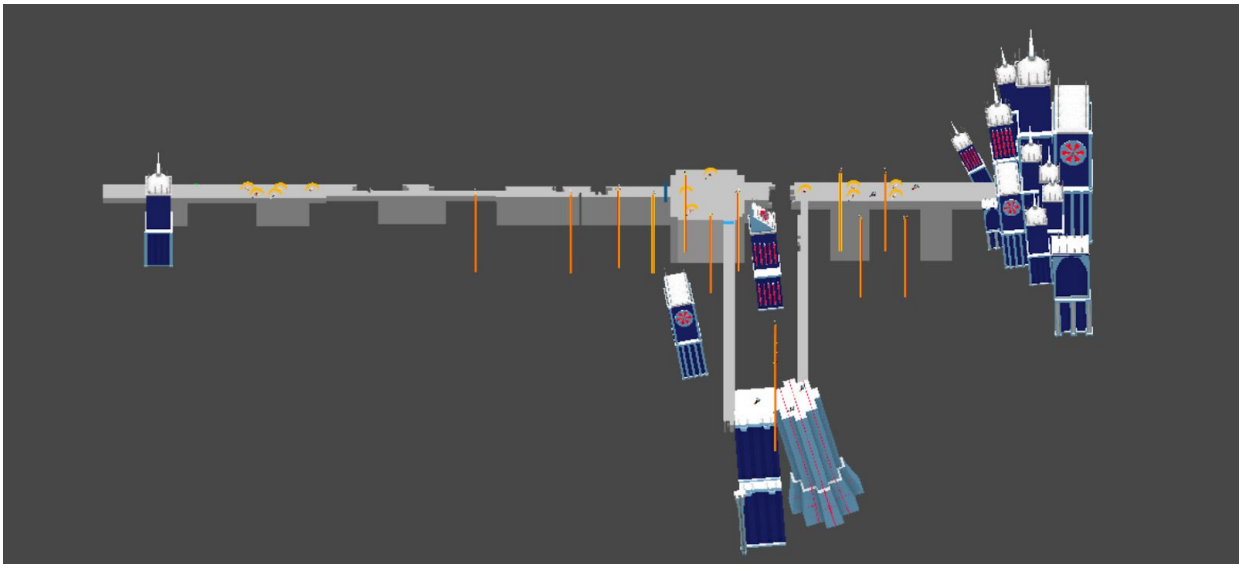
Tercer nivel con enemigos

- Se juega el nivel (tanto internamente como con personas ajenas al proyecto), se comprueban sus puntos fuertes y débiles, y se itera debidamente (cambiando la distribución o el número de enemigos, el trazado del mapa, etc)
- Se incorporan *placeholders* de los elementos del nivel (edificios, elementos decorativos, ect) para comprobar el efecto estético



Tecer nivel con *placeholders*

- Si el resultado es satisfactorio, el nivel se cierra. Una vez en la fase correspondiente, y con todos los niveles ya generados, se procede a sustituir los *placeholders* por el arte final
- Se hace un último testeo, y se realizan los ajustes finales



Tercer nivel con arte definitivo

4. Gestión del proyecto

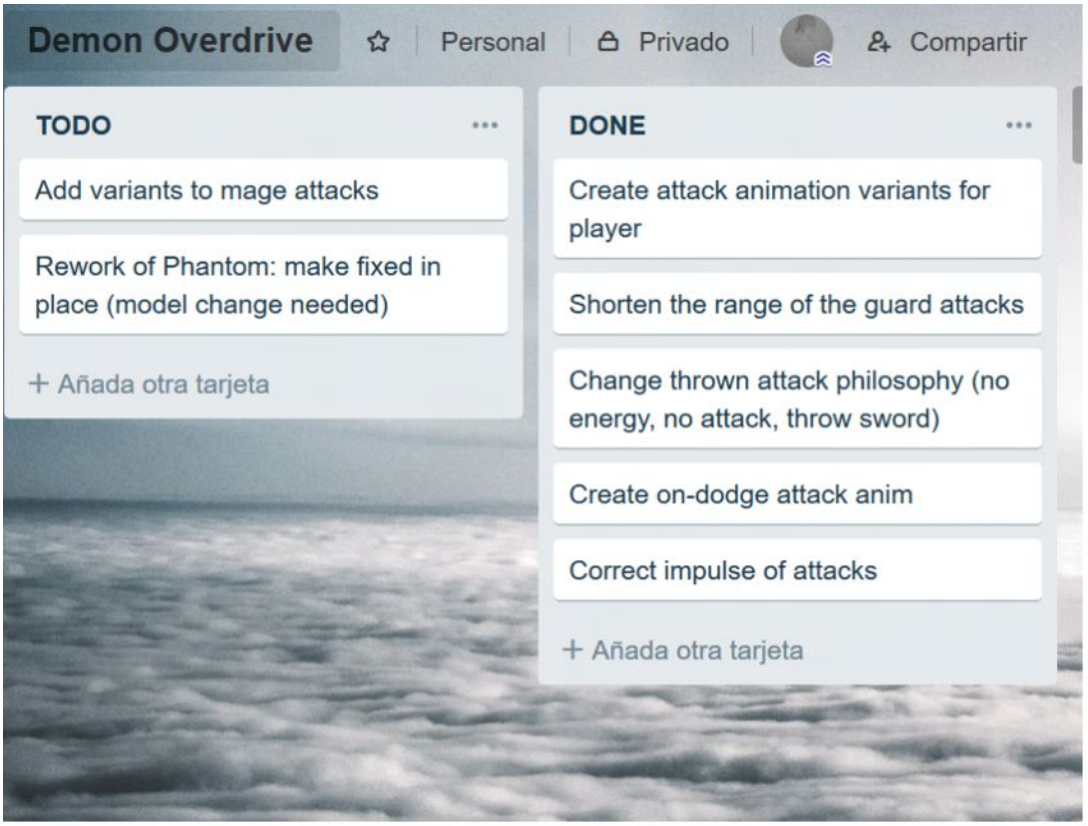
4.1 Procedimiento

4.1.1 Calendario

Debido a la naturaleza unipersonal del proyecto, no es indispensable detallar al máximo las tareas e interdependencias del desarrollo (ya que no hay una necesidad de comunicación detallada entre distintos departamentos). Sin embargo, conviene planificar y asignar tiempo a las tareas a realizar, aunque sea de manera general. Por este motivo, en lugar de un diagrama de Gantt, se ha optado por usar un calendario tradicional. De esta manera, tenemos el proyecto ordenado y planificado, pero también disponemos de bastante flexibilidad en caso de contingencias o imprevistos.
Cada segmento de color corresponde a un *sprint*, o período donde una determinada tarea debe ser finalizada.

4.1.2 Trello

Una vez más, debido a ser un proyecto unipersonal, se han rechazado herramientas complejas (como Jira o Hack and Plan), más orientadas a grupos numerosos, en favor de la inmediatez y sencillez de Trello.



Tablero de Trello de *Demon Overdrive*

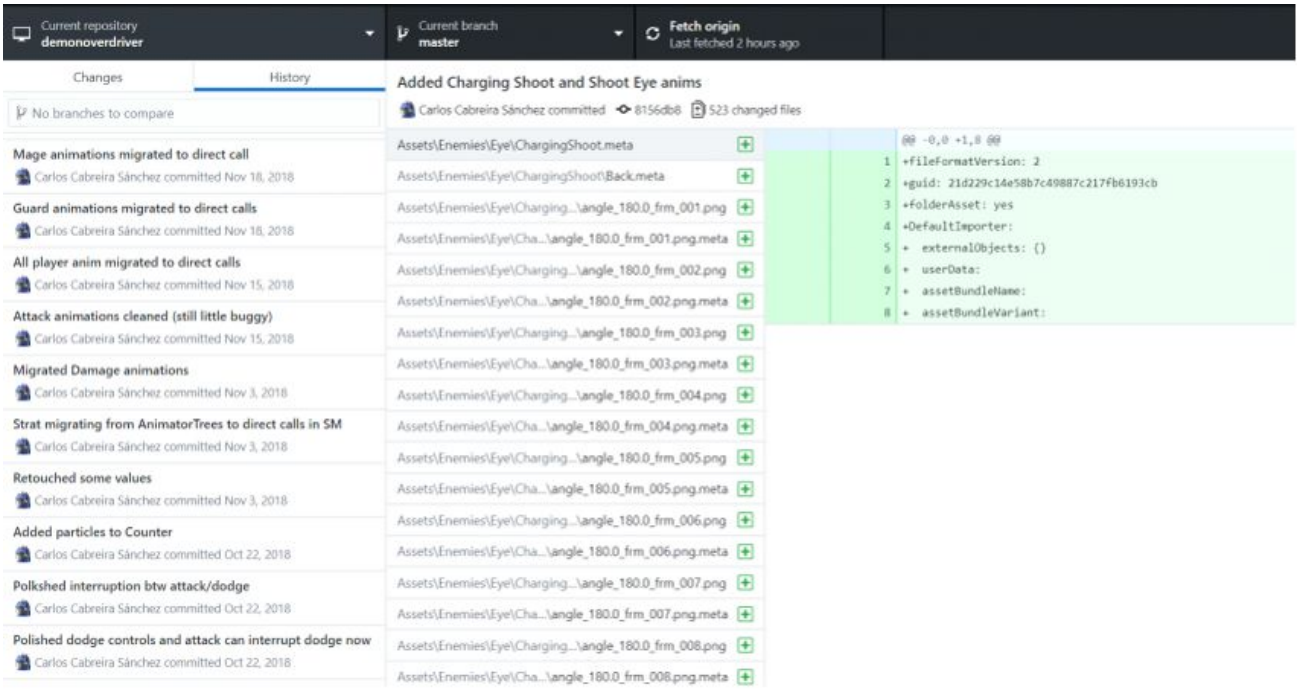
Las diversas tareas se apuntan en la columna TODO, y se mueven a DONE una vez realizadas. Es una versión muy simplificada de la metodología *Kanban*, y se usa más para recordatorio personal que como herramienta de comunicación (por los motivos expuestos anteriormente).

4.1.3 GitHub y control de versiones

Para este proyecto, se usa una combinación de BitBucket y Github Desktop. Debido al uso de material de pago en el desarrollo del proyecto, el código requiere de ser privado (razón por la cual no es posible enlazar el repositorio aquí), y BitBucket es de los pocos repositorios que permite esto de manera gratuita.

Github Desktop fue escogido frente a otras aplicaciones de escritorio(como SourceTree) por su sencillez e inmediatez.

El procedimiento es el siguiente: tras implementar un cambio mínimamente sustancial (programar una mecánica, añadir arte, o modificar un nivel, por ejemplo) y comprobar su funcionamiento, se «cierra» esa versión del proyecto (lo que se denomina «hacer un *commit*») en Github Desktop. Tras ponerle un nombre descriptivo, ese *commit* se sincroniza y se almacena de manera online en BitBucket. De esta manera, se tiene acceso remoto al proyecto en diferentes estados de desarrollo, ordenados de manera cronológica.



Repositorio de *Demon Overdrive* en Github Dekstop

Dichas versiones pueden ser recuperadas más adelante por cualquier motivo (se daña el equipo, se corrompe algún archivo, o simplemente se quiere recuperar una característica que se hubiera eliminado). Huelga decir que, cuanto menos tiempo pase entre commit y commit, más seguro estará el proyecto.

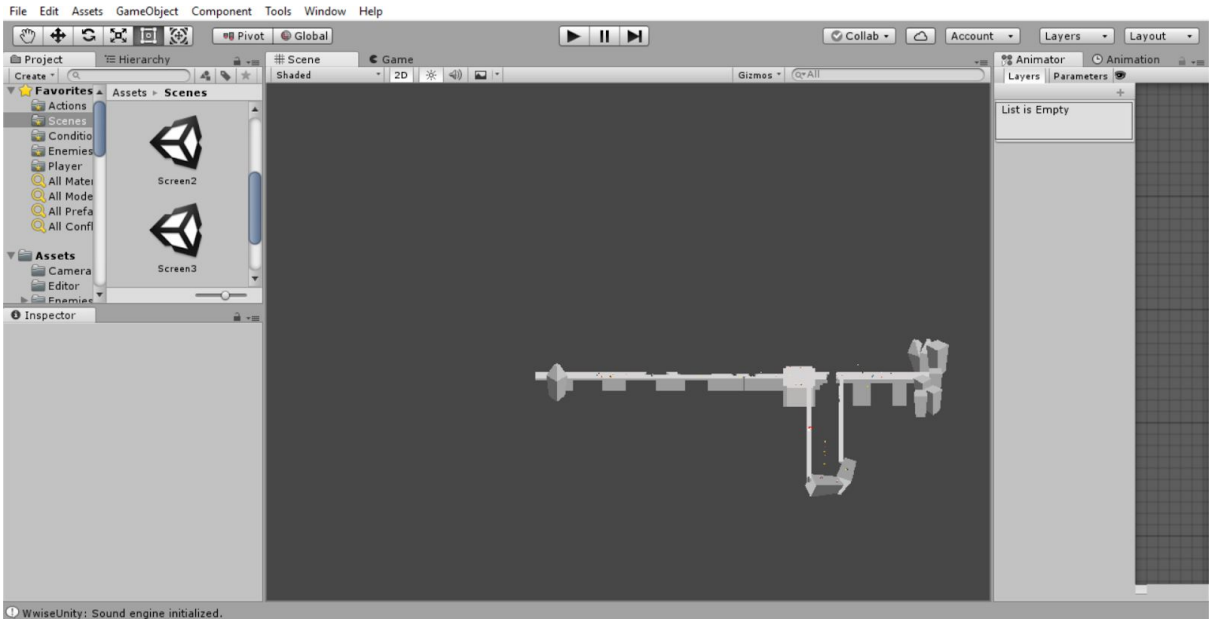
Cada vez que se alcance una versión estable del proyecto con un cambio significativo (un nuevo enemigo o un nuevo nivel, por ejemplo), se hará un ejecutable (a partir de ahora, *Build* o *Release*) que se testeará, para certificar su estabilidad.
En los nombres de las releases figurará el nombre *Overdrive*, seguido del número de versión:

- Las releases creadas durante preproducción tendran números entre 0.0.1 y 0.0.9 (prototipo final).
- Las releases creadas durante producción irán desde la 0.1.0 hasta la 0.7.0 (Alpha)
- Las releases de postproducción irán desde la 0.7.1 hasta la 0.9.0 (Beta)
- La versión final del juego (Gold) será la 1.0.0

4.2 Herramientas y programas

4.2.1 Unity

Para el desarrollo del juego propiamente dicho, se ha optado por el motor de juegos (o *Game Engine*) Unity, por las siguientes razones:
Es gratuito en su versión no comercial
Es intuitivo y fácil de manejar (algo necesario, debido al estricto límite de tiempo del proyecto)
Dispone de una amplia documentación y comunidad en línea, a la que es fácil acceder en caso de necesidad

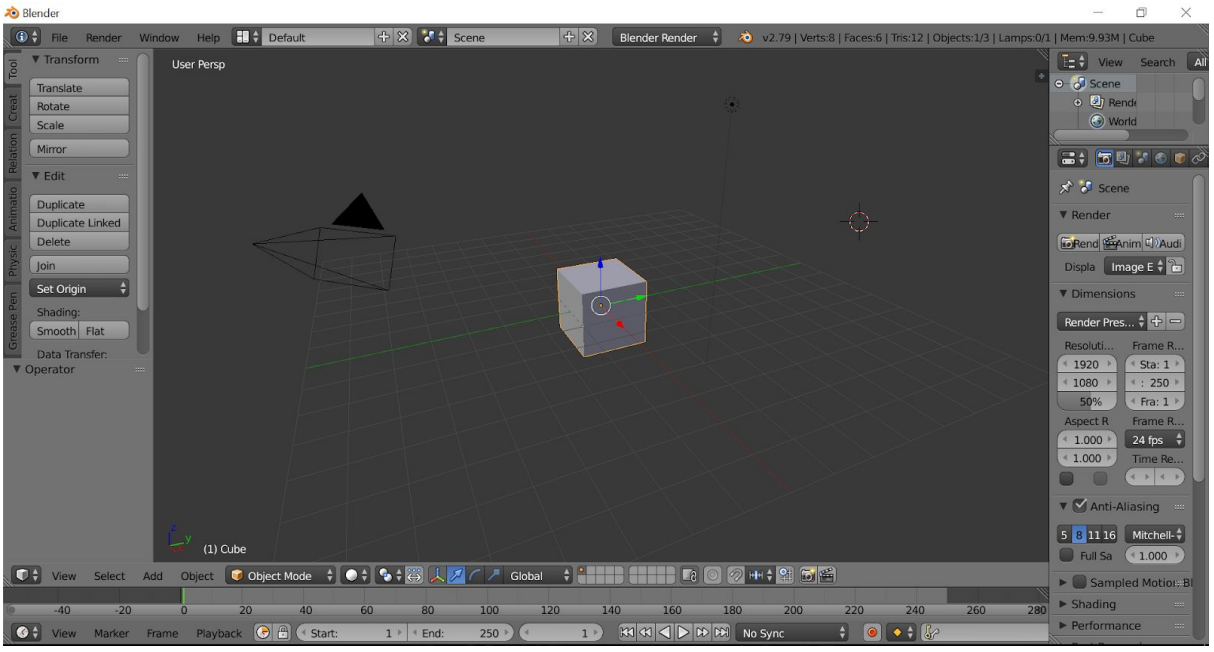


Captura de pantalla de Unity

4.2.2 Blender

Para la creación de los modelos 3D, se ha optado por la herramienta Blender, por los siguientes motivos:

- Es gratuita y de código abierto
- El ligera y rápida de manejar
- Tiene todas las funcionalidades que este proyecto precisa (herramientas de modelaje, animación y procesamiento de imagen)
- Permite crear código en lenguaje Python (tal y como se ha mostrado anteriormente en esta memoria, en el apartado 3: Metodología), para automatizar ciertas tareas.

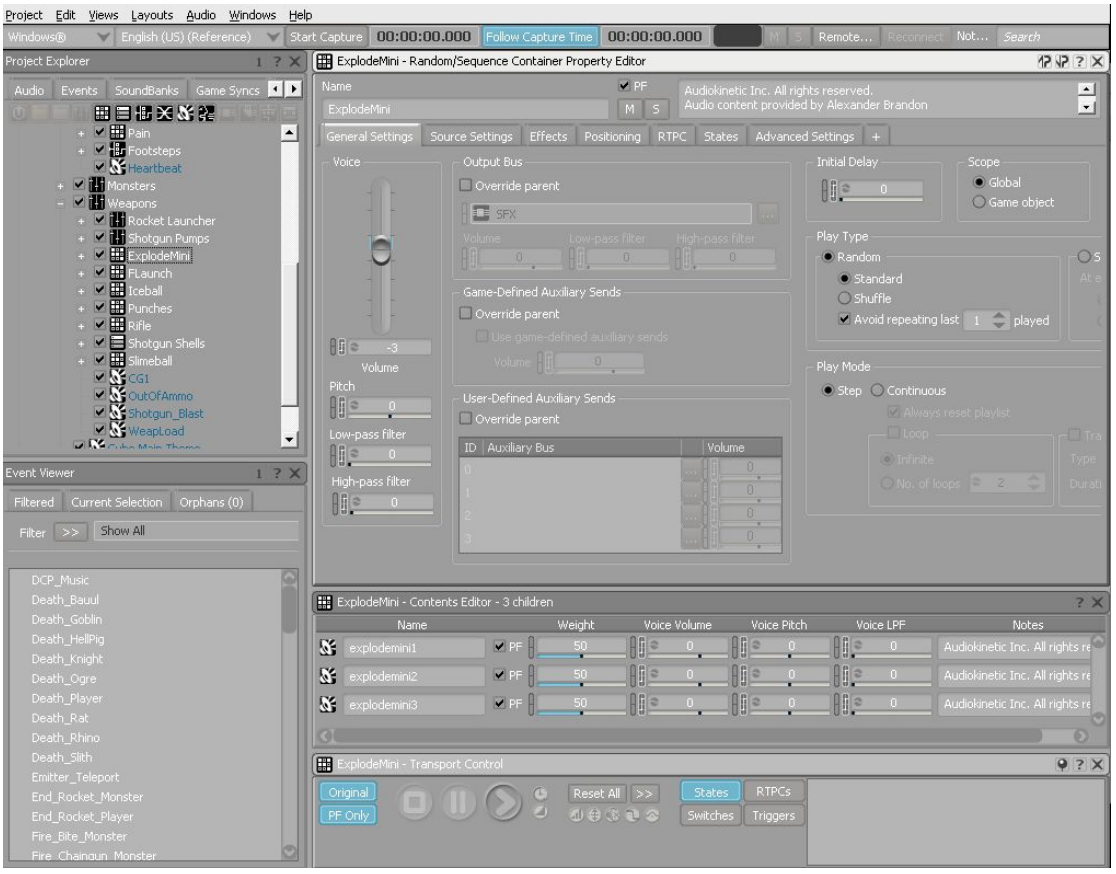


Pantalla de bienvenida de Blender

4.2.3 Wwise y tratamiento de audio

Para manejar el audio del juego, se ha escogido el motor de audio Wwise, por encima del integrado en Unity, por los siguientes motivos:

- Es gratuito para proyectos no comerciales (con un límite de 200 archivos de audio).
- Permite hacer, de manera sencilla, efectos sonoros complejos (transiciones, reverberaciones y mezclas).
- Es intuitivo y ofrece mucho control sobre el tratamiento de los archivos.
- Dispone de integración directa en Unity.



Captura de pantalla de Wwise

Respecto a los archivos de audio: estos serán los únicos elementos del proyecto que no serán creados internamente, sino que se recurrirá a páginas de recursos sonoros gratuitas, como FreeFX o FreeSound debido, una vez más, a la limitación de tiempo y personal.
La música será compuesta por Guillermo Otero Costas, músico e historiador gallego.

4.3 Métodos de validación

Los principales métodos de validación serán los testeos o tests, y habrá de dos tipos: internos y externos.
Los internos serán llevados a cabo por el desarrollador de este proyecto, y serán realizados después de cada release. El objetivo de estos tests es comprobar el correcto funcionamiento de la aplicación después de añadirse contenido nuevo; registrar todos los posibles errores y clasificarlos según su urgencia; y comprobar que el contenido añadido alcanza las expectativas. Según los resultados de estos test, puede suceder lo siguiente:

- Todo funciona según lo previsto, o los bugs hallados son leves, y se procede al siguiente sprint.
- Surgen bug severos que requieren de atención inmediata. Se procede a un período de *bugfixing*, que tendrá prioridad sobre el siguiente *sprint*.
- El contenido añadido no alcanza las expectativas. En caso de no haber alcanzado la fecha límite del sprint actual, se procede a rehacer e iterar el contenido. En caso de que se alcance la fecha límite del sprint , y dicho contenido siga por debajo del estándar de calidad, se descartará (atendiendo siempre a la prioridad de dicho contenido, detallada en el apartado *Plan de contingencias*, que figura más abajo).

Los test externos están destinados a valorar la experiencia de usuario, y se comprobará la opinión y reacciones del usuario respecto a:

- Controles
- Comodidad
- Entendimiento del juego
- Valoración del nuevo contenido (si el mismo usuario ha participado en tests anteriormente)
- Valoración general del juego y su disfrute
- Posibles problemas y mejoras.

Los tests externos están planeados para las siguientes releases:

- 0.0.5, 0.0.7 y 0.0.9 (Preproducción)
- 0.4 y 0.7 (Alpha)
- 0.9 (Beta) y 1.0 (Gold)

Si el resultado general de estos tests es favorable, se procede a avanzar al siguiente sprint.
En caso contrario, se valorarán los resultados obtenidos y se actuará en consecuencia (por ejemplo, si una mecánica o enemigo no gustan, se eliminarán o modificarán), siempre respetando el calendario y el plan de contingencias.

4.4 DAFO

Este gráfico se elaboró al finalizar la fase de preproducción, una vez creado un prototipo jugable, gracias al cual se adquirió suficiente perspectiva y contexto.
Esta tabla está basada en una hipotética comercialización de *Demon Overdrive*, en caso de desarrollarse como juego completo.

Fortalezas: *Gameplay* inmediato y divertido. Desarrollo eficiente gracias a la técnica empleada.	**Debilidades:** Equipo unipersonal. Experiencia media/baja en el desarrollo.
Oportunidades: Nicho de mercado casi vacío	**Amenazas:** Tiempo limitado. Marketing insuficiente. La estética derivada del *workflow 3D-2D* no convence a los jugadores.

4.5 Riesgos y plan de contingencias

El principal riesgo de este proyecto es que se alcancen las fechas de entrega sin haber logrado el objetivo marcado en el *sprint*. Debido a ello, se han organizado las tareas por prioridad. De menor a mayor:

Generación de contenido
La creación de enemigos y niveles está marcada claramente en el calendario. En caso de que no se consiguiera finalizar alguno a tiempo, podría ocurrir:
Si el *sprint* siguiente está reservado para crear contenido equivalente (por ejemplo, otro enemigo), en vez de eso, se asignará la finalización de la tarea inacabada, descartando el contenido a crear.
En caso contrario, el contenido sin finalizar se descartará.

La creación de contenido ***no tendrá nunca mayor prioridad que el resto de las tareas.***

Audio y arte
El arte tiene también su *sprint* asignado en el calendario. Todo el arte debe ser finalizado en ese tiempo, aunque sea en una versión primitiva (***pero aceptable***). Los *placeholders* que fueron añadidos durante la creación de los niveles deberán ser todos sustituidos por arte final.
Existe también el riesgo de que Otero Costas, el músico asociado, no produzca las pistas de audio a tiempo, o que las produzca con una calidad inaceptable. En este caso, la música del juego se extraería de fuentes gratuitas como el canal NoCopyrightMusic, donde se dispone de diverso material gratuito.

Pulido (durante post-producción)
Durante la fase de pulido no se generará contenido nuevo (niveles, enemigos, mecánicas o arte). Los esfuerzos se centrarán en mejorar lo ya existente (ejemplo, añadir partículas, realizar correcciones de color, efectos de post-procesado, corregir velocidades y animaciones, etc). ***Se priorizará la calidad del contenido respecto a la cantidad***.

Bugfixing
Esta es la tarea de mayor prioridad sobre todas. Su fecha de inicio es inamovible (salvo para adelantarla), y finalizará cuando no queden registrados *bugs* de frecuencia media o superior.

4.6 Análisis inicial de costes

Estimación de costes del desarrollo de Demon Overdrive

GASTOS REALES				
Gasto	**Tipo**	**Coste**	**Años de amortización**	**Coste Total**
Mantenimiento personal	Mensual	1.000,00 €		10.000,00 €
Alquiler	Mensual	300,00 €		3.000,00 €
Luz y agua	Mensual	50,00 €		500,00 €
Computadora (portátil)	Amortización	1.000,00 €	6	138,89 €
Mobiliario	Amortización	300,00 €	6	41,67 €
Auriculares	Amortización	30,00 €	4	6,25 €
Visual Studio (Community)	Gratuito	0,00 €		0,00 €
Photoshop	Mensual	21,00 €		210,00 €
Bitbucket/Github Desktop Personal	Gratuito	0,00 €		0,00 €
Unity Personal	Gratuito	0,00 €		0,00 €
Sonidos y música *freeware*	Gratuito	0,00 €		0,00 €
Duración del proyecto (meses):				
10			TOTAL	**13.896,81 €**

Estimación de costes hipotéticos, en caso de comercializarse

Nótese que todas las licencias de herramientas son gratuitas, debido a la naturaleza unipersonal del proyecto. En caso de tratarse de un equipo de mayor tamaño, los costes variarían.

GASTOS HIPOTÉTICOS				
Gasto	**Tipo**	**Coste**	**Años de amortización**	**Coste Total**
Mantenimiento personal	Mensual	1.000,00 €		10.000,00 €
Alquiler	Mensual	300,00 €		3.000,00 €
Luz y agua	Mensual	50,00 €		500,00 €
Computadora (portátil)	Amortización	1.000,00 €	6	138,89 €
Mobiliario	Amortización	300,00 €	6	41,67 €
Auriculares	Amortización	30,00 €	4	6,25 €
Visual Studio (Community)	Gratuito	0,00 €		0,00 €
Photoshop	Mensual	21,00 €		210,00 €
Bitbucket/Github Desktop Personal	Gratuito	0,00 €		0,00 €
Unity Personal	Gratuito	0,00 €		0,00 €
Músico	Mensual	2.000,00 €		20.000,00 €
Duración del proyecto (meses):				
10			TOTAL	**33.896,81 €**

En caso de comercializarse (en versión exclusivamente digital, con un 70% de retención por copia), estas serían las estimaciones de ventas:

	Equilibrio	Escenario favorable	Escenario muy favorable
Copias Vendidas	3.229	10.000	20.000
Precio por copia	15,00 €	15,00 €	15,00 €
Ingresos Brutos	48.435,00 €	150.000,00 €	300.000,00 €
Ingresos Netos	33.904,50 €	105.000,00 €	210.000,00 €
Costos Totales	33.896,81 €	33.896,81 €	33.896,81 €
Beneficios	7,69 €	71.103,19 €	176.103,19 €

Ingreso por copia
70,00 %

Se ha optado por un precio de 15 euros por copia, siendo este el precio medio de un juego independiente en diversas tiendas (entre 10 y 20 euros).

5. Desarrollo

5.1 Preproducción

La preproducción de *Demon Overdrive* se inició el 1 de septiembre de 2018 y finalizó el 20 de febrero de 2019.
Los objetivos que se deberían alcanzar durante preproducción eran:

- Establecer las mecánicas del juego, después de numerosas iteraciones.
- Establecer los pasos a seguir para la creación de enemigos y escenarios.
- Experimentar con diferentes estilos gráficos y, posteriormente, apegarse a uno.
- Establecer el ritmo del juego: velocidad de animaciones y personajes, cadencias de ataques y respuesta emocional del usuario.
- Establecer los patrones de programación que se seguirán dentro de Unity para los diferentes sistemas.

Todos y cada uno de ellos fueron cumplidos con holgura.

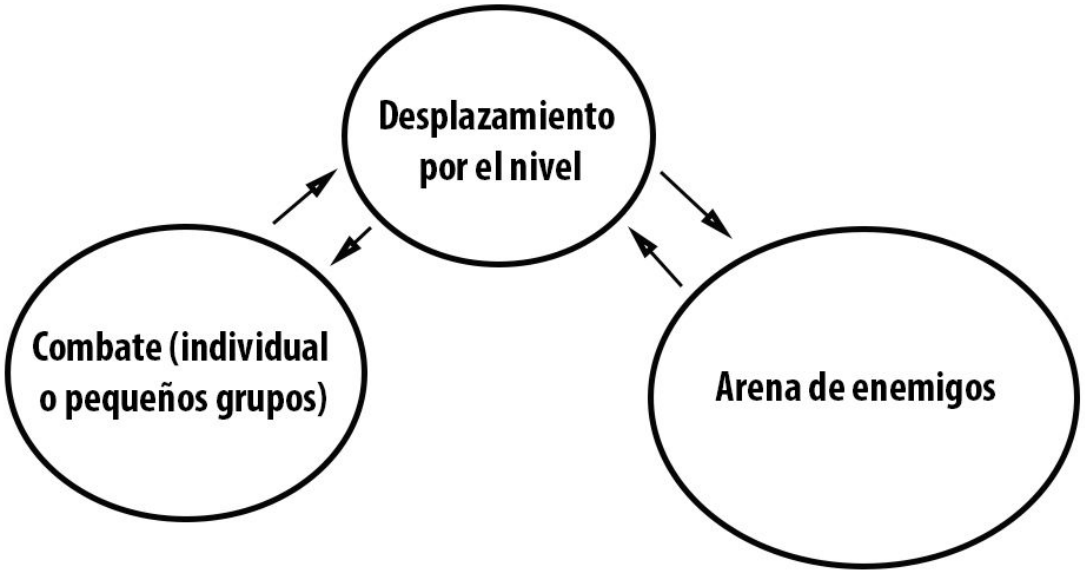
5.1.1 Establecimiento del ritmo del juego y respuesta emocional

Dado que este proyecto se inspira en el previamente comentado género *hack and slash*, es de esperar que comparta muchos elementos de títulos como *NieR, Bayonetta* o *Devil May Cry,* a pesar de las diferencias en su presentación.

El aspecto más característico de este género es su *alta cadencia de ataque*. El jugador suele realizar varios movimientos ofensivos por segundo, dando lugar a un *gameplay* notablemente veloz. Siguiendo con esta filosofía, estos juegos *suelen recompensar a los jugadores precisos*, ya sea con combos espectaculares (*Bayonetta)* o con una mayor puntuación al final de la partida.
Asimismo, es común que el jugador se encuentre en una *remarcable desventaja numérica* (para compensarlo, cada enemigo no suele aguantar más de unos pocos ataques). Esto sirve, unido a lo mencionado en el párrafo anterior, para generar una *fantasía de poder*, en la que el jugador se recree.

Es recomendable remarcar que una fantasía de poder no requiere que el juego sea necesariamente fácil.

El ciclo de acciones que se quiere conseguir en el jugador es el siguiente:



Ciclo de acciones núcleo

Desplazamiento por el nivel: decaimiento del ritmo, descanso entre batallas, avance

Combate con enemigos individuales (o grupos pequeños): incremento de la tensión, anticipación antes de los retos principales. Existe la opción de pasar de largo.

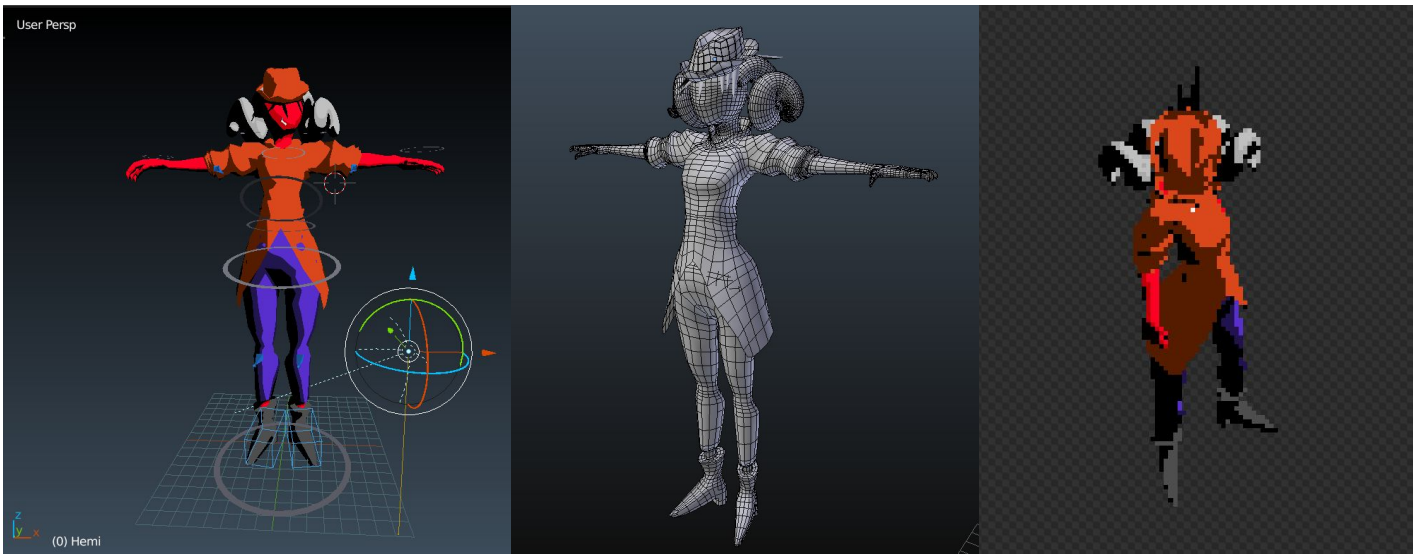
Arena de enemigos (grupos grandes): concentración del reto, pico de dificultad y ritmo dentro del nivel.

Por otro lado, se pretende organizar los dos tipos de retos (enfrentamientos pequeños y grandes) de manera que cada uno se perceptiblemente más difícil que el anterior.

5.1.2 Creación del personaje jugador y sus mecánicas

Tras establecer las bases de la creación de personajes (cuyos detalles se pueden ver en el apartado 3.2 - Creación de personajes con el *workflow 3D*), se procedió a crear el personaje que encarnaría el jugador, conocida dentro del juego como "*The Overdriver*". Unos bocetos iniciales en papel (de los que se hicieron varias iteraciones) sirvieron como punto de partida para, posteriormente, crear un modelo 3D (arte no definitivo).

Bocetos de la protagonista de Demon Overdrive



Modelo primitivo de la protagonista

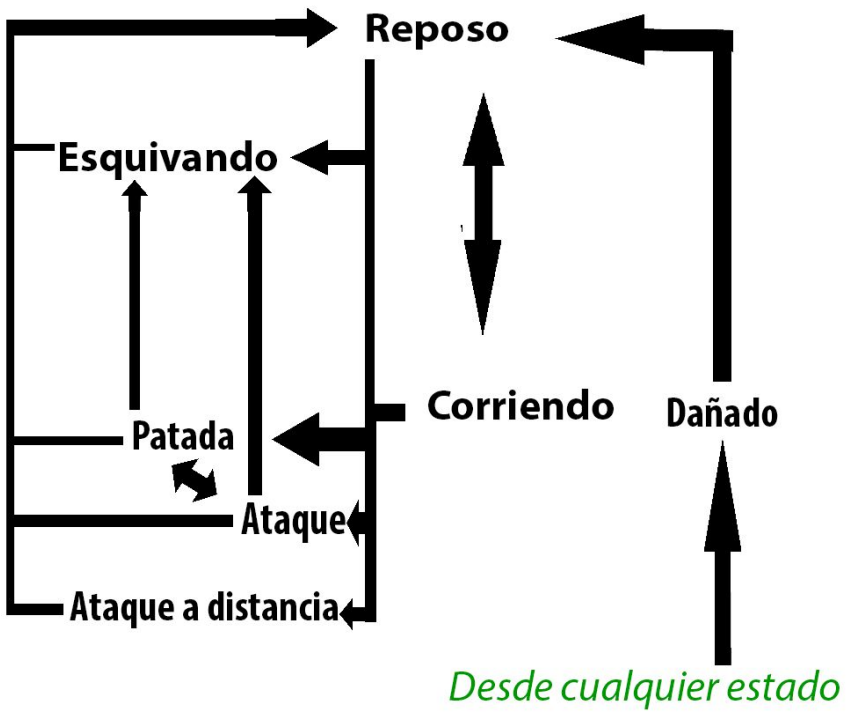
Modelo definitivo de la protagonista (realizado durante post-producción)

Una vez obtenidos los fotogramas de animaciones básicas (como la de reposo o la del ciclo de caminar), se procedió a diseñar y programar al jugador dentro de Unity.

Para programar a la protagonista (y, posteriormente, a los enemigos), se optó por usar un patrón de *Máquinas de Estado Finitas* (*Finite State Machines* o *FSM,* en inglés).
Este patrón (construído en el lenguaje C# que integra Unity) consiste en asignarle al objeto en cuestión un estado, que determina un comportamiento específico (por ejemplo, si está quieto, si está corriendo, etc). El objeto *solo puede estar en un estado a la vez*, y las transiciones entre estados vienen condicionadas por el programador (de esta forma, se puede controlar desde que estados a que estados se puede transicionar, y con qué condiciones).

Además de un gran control y claridad, este patrón de programación facilita asignar y eliminar comportamientos concretos, facilitando mucho la iteración.

La máquina de estados del personaje jugador de *Demon Overdrive*, al momento de finalizar preproducción, tenía este aspecto:



Máquina de estado del personaje jugador

Una vez creados los dos estados más básicos (*En reposo* y *Corriendo)* y sus animaciones correspondientes, se procedió a diseñar el resto de mecánicas. Siguiendo los pilares del juego, dichas mecánicas deberían:

A) Focalizarse en el combate (dialogo, uso de objetos y similares quedan automaticamente descartados)
B) Centrarse en el movimiento

Dado esto, mecánicas de ataque y defensa suponían el punto de partida más lógico.

Ataque básico

La mecánica principal, el ataque, fue objeto de numerosas iteraciones a lo largo de preproducción, aunque su filosofía permaneció inalterable: un ataque rápido, cuerpo a cuerpo, que desplazara al jugador ligeramente hacia adelante, y que se pudiera encadenar con otras acciones.

Fotograma: ataque

Se partió de un esquema en el que el jugador atacaba en diferentes direcciones con diferentes botones (en un mando de Xbox, arriba con Y, abajo con A, izquierda con X y derecha con B). Sin embargo, fue rápidamente descartado, debido a la incomodidad que suponía para muchos jugadores y, sobre todo, a la innecesaria complejidad añadida. Finalmente, se migró a un esquema de control de atacar con un unico boton y usar los controles de movimiento para elegir la dirección.

Estéticamente, el ataque también evolucionó: la animación, en un principio, mostraba a la protagonista atacando a manos vacías. Tras varios testeos con jugadores externos, se añadió un arma (una enorme espada), para facilitar la visualización del ataque por parte del jugador.

Esquive

El esquive (consistente en un desplazamiento veloz y un breve tiempo de invulnerabilidad) sustituyó al bloqueo como opción defensiva, ya que se adhiera mejor a los pilares básicos del juego (en especial, al de desplazamiento).

Fotograma: esquive

La mecánica en sí no sufrió de demasiados cambios en su concepto (dada la alta satisfacción expresada por los sujetos de playtesting): tan solo se iteraron valores numéricos como la velocidad o la distancia del esquive.

Mecánicas Secundarias: Diseño

Una vez establecidos los movimientos básicos, se procedió a un proceso de iteración, donde se crearon y testearon las que serían mecánicas secundarias. Para ser catalogadas como “listas para producción”, deberían cumplir los siguientes requisitos (además de respetar los pilares del juego):

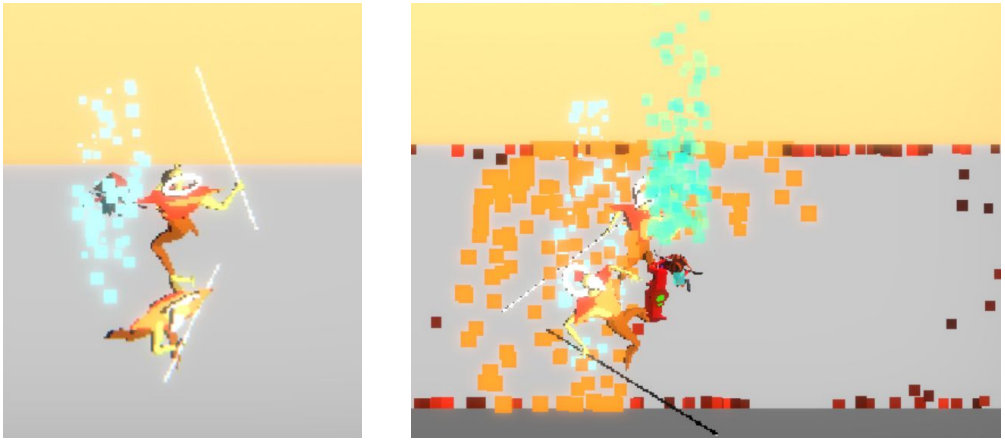
- **Responder a una necesidad**: proveer al jugador de una herramienta para solucionar situaciones determinadas
- **Doble filo**: por cada ventaja, un inconveniente, para tratar de evitar que haya mecánicas claramente mejores que otras
- **Doble función**: aparte de la función principal de la mecánica, esta deberá tener un uso secundario de manera que, empleada en situaciones diferentes, desemboque en resultados diferentes (cosa que ampliaría la profundidad del gameplay sin necesidad de generar mas contenido)..
- **Sencillas de entender y desarrollar**
- **Divertidas**: la mecánica en cuestión deberá ser revisada, testeada y aprobada por jugadores externos.

Mecánicas secundarias aprobadas (y por que se aprobaron)

- **Patada** (*lanza al enemigo. Si choca contra una pared, le causa daño*): visualmente atractiva, enfatiza el movimiento, y otorga la capacidad de alejar a un enemigo, útil cuando el jugador está rodeado.

Fotograma: patada

- **Contraataque** (*si el jugador esquiva justo cuando impacta un ataque, lo devuelve, causando daño doble en una pequeña área*): recompensa el juego preciso, visualmente atractivo, sirve tanto para ataque como para defensa.
- **Teletransporte** (*justo después de dar una patada a un enemigo, el jugador puede teletransportarse a él y causar doble daño en una pequeña área*): enfatiza mucho el movimiento, recompensa el juego preciso, ayuda a escapar de zonas peligrosas.



Fotogramas: Esquive/ Contraataque + Teletransporte

- **Arrojar y recuperar la espada** (*ataque a distancia que causa daño tanto a la ida como a la vuelta. La vuelta no tiene por que ser inmediata*): da opciones de ataque al jugador y permite atraer enemigos, pero le impide atacar mientras no recupere la espada.

Fotograma: arrojar la espada

Mecánicas secundarias descartadas (y por que se descartaron):

- **Ataque en área** (*golpe al suelo que causaba daño y empujaba a enemigos en un área circular alrededor del jugador. Requeria energía, que se obtenía golpeando enemigos*)**:** no cumplia con el pilar de “siempre en movimiento”, el medidor de energía suponía una complejidad innecesaria, y su uso se solapaba con el teletransporte y el contraataque.
- **Ataque en carga** (*ataque en movimiento hacia adelante, realizado después de esquivar):* no tenía desventajas ni aplicaciones secundarias. Poco interesante.
- **Encadenamiento de teletransportes** (*tras un ataque en teletransporte, se podía realizar de nuevo de manera inmediata si se era suficientemente preciso):* si bien visualmente espectacular, resultaba demasiado ventajoso, y no tenía aplicaciones secundarias. No respondía a ninguna necesidad.

5.1.3 Creación de los primeros enemigos

El proceso de creación de enemigos es muy similar al de la protagonista: se sigue el mismo patrón de modelado y programación (máquinas de estado finitas).

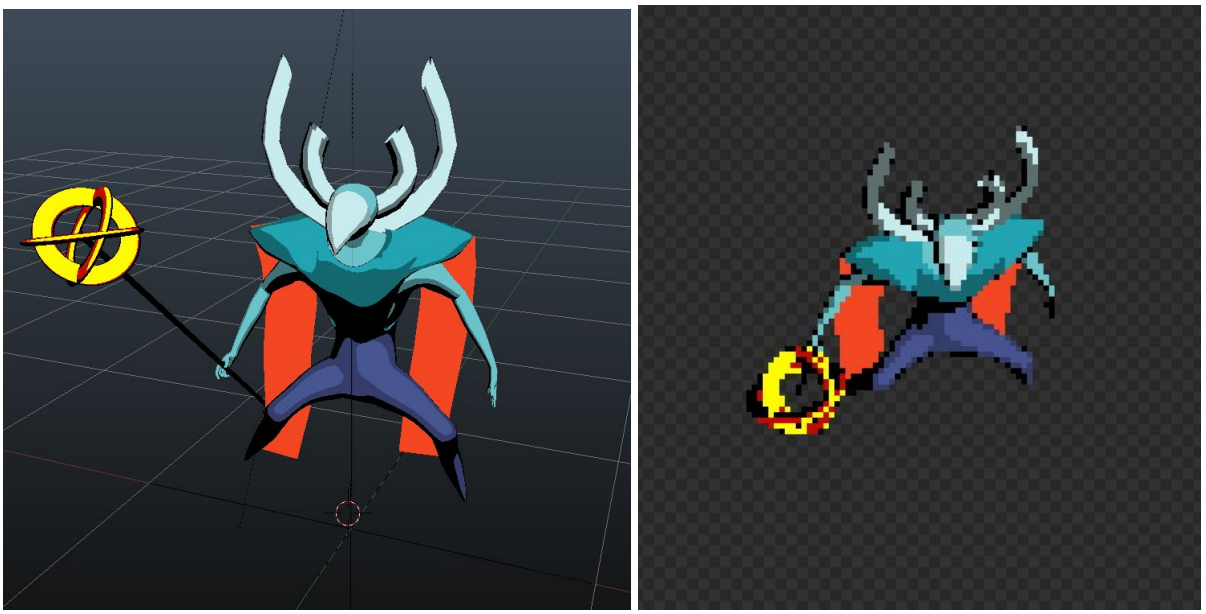
Los fundamentos de diseño si son propios. Cada enemigo debe:

- Ser facilmente reconocible, tanto en forma como en color.
- Tener un único comportamiento, diferenciado del resto de enemigos
- Exhortar al jugador a usar los movimientos a su disposición, incentivando el uso de distintas mecánicas
- Afectar al espacio de manera propia, para generar nuevos patrones de movimiento en el jugador

El principal plan durante preproducción era el de desarrollar un solo enemigo, para probar diferentes aspectos de su desarrollo, antes de generar más contenido. Dicho esto, se creó al **Guardia**, el enemigo más común y sencillo de *Demon Overdrive*. Su comportamiento se limita a acercarse al jugador y atacarle a corta distancia.

Guardia (modelo y sprite)

Sin embargo, para poder testear las distintas mecánicas, en especial las de largo alcance, se requería un tipo de enemigo distinto, que complementase al Guardia. Identificada esta necesidad, se implementó un segundo enemigo, el **Mago**, que ataca con proyectiles destruibles.



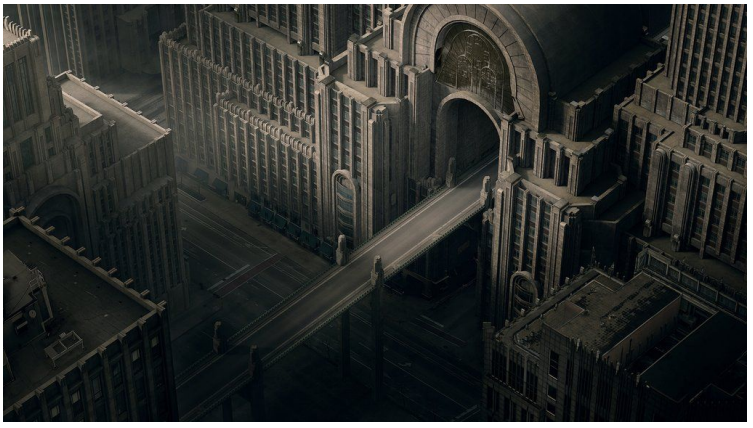
Mago (modelo y sprite)

Gracias a la complementación de dos unidades tan dispares, se pudo poner en práctica todas las intenciones de diseño y las distintas mecánicas del jugador.

5.1.4 Creación de escenarios: arte y método

Durante preproducción, se barajaron diversas opciones para crear los escenarios.

Lo primero que se debía definir era la ambientación y estética. *Demon Overdrive* transcurre en una ciudad donde está sucediendo un conflicto.
Las principales sensaciones que se querían provocar en el jugador eran las de *ominosidad* y *escala*.
Tras varias sesiones de investigación, se encontraron dos estilos arquitectonicos que cumplian con este requisito: el estilo **gótico,** propio de los edificios religiosos del renacimiento, y que se asocia a la grandiosidad y la trascendencia; y el **art deco,** recurrente a principios del siglo XX, muy apegado a la gran ciudad, las grandes escalas y el progreso imparable



Referencias de los dos estilos artísticos de Demon Overdrive

La combinación de ambos estilos en uno solo parecía ajustarse perfectamente a la estética urbana gigantesca e intimidante que se estaba buscando.

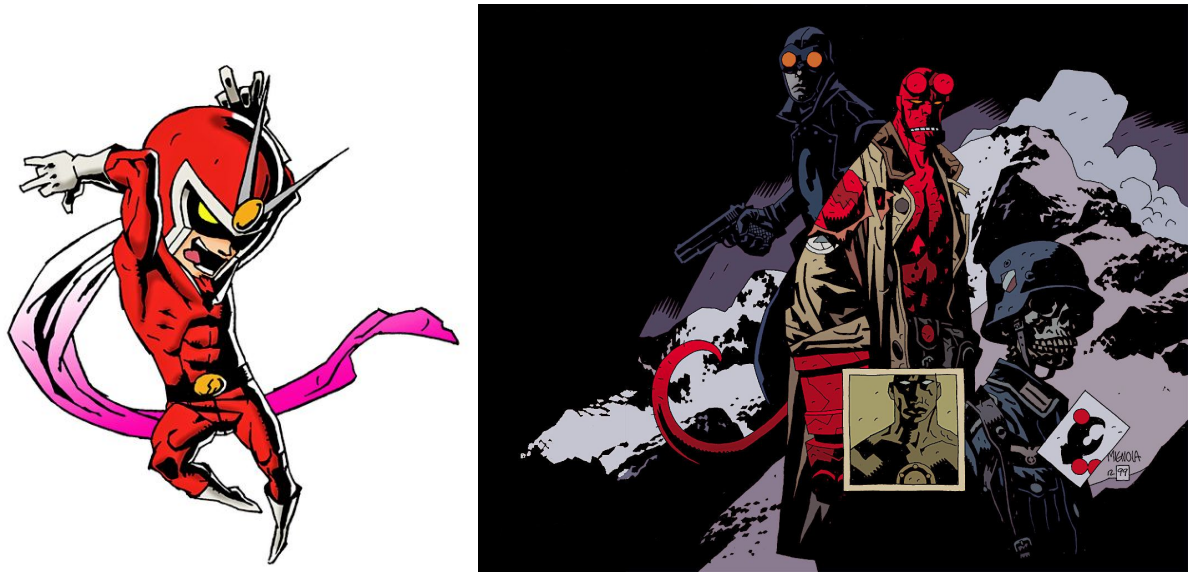
Se plantearon dos métodos para la creación de los escenarios:

- Emplear la herramienta de Tileset integrada en Unity. Esta herramienta basa el escenario en una cuadrícula, donde el usuario va “pintando”, como si de un mosaico se tratara. La principal ventaja de este método es que el resultado final se controla mucho mejor. Por contra, resulta lento.
- Adaptar el *workflow 3D* usado en la creación de personajes. Se generarían diversas plantillas de edificios y demás elementos en Blender, que después se renderizarían y exportarían a Unity. Una vez aqui, es cuestion de posicionar y adaptar diversas instancias de los elementos, como si de una maqueta se tratase. Es un proceso mucho más rápido que el anterior, pero es difícil controlar el resultado (en especial, los detalles pequeños).

Después de probar ambos, se ha optado por usar el *workflow 3D* (debido a su rapidez y a que es un contexto perfecto para poner a prueba la técnica), aunque no se descarta usar el Tileset para suelos y similares.

5.1.5 Experimentacion artística

El *workflow 3D* condiciona en gran medida el estilo artístico.
Los sombreados realistas se descartaron rápidamente. La falta de detalle daba un estilo “sucio” a los sprites, muy similar a los antiguos modelos prerrenderizados de juegos como Donkey Kong Country. Por este motivo, colores planos y transiciones duras fueron el camino a seguir.
Para añadir un toque de personalidad, la paleta de colores se ha creado en similitud a la estética del cómic *Hellboy* (Mike Mignola, 1993-2016) y el juego *Viewtiful Joe* (Team Viewtiful, 2003): sombras de negro absoluto contrastadas con colores saturados y vivos.



Viewtiful Joe (derecha) y Hellboy (izquierda)

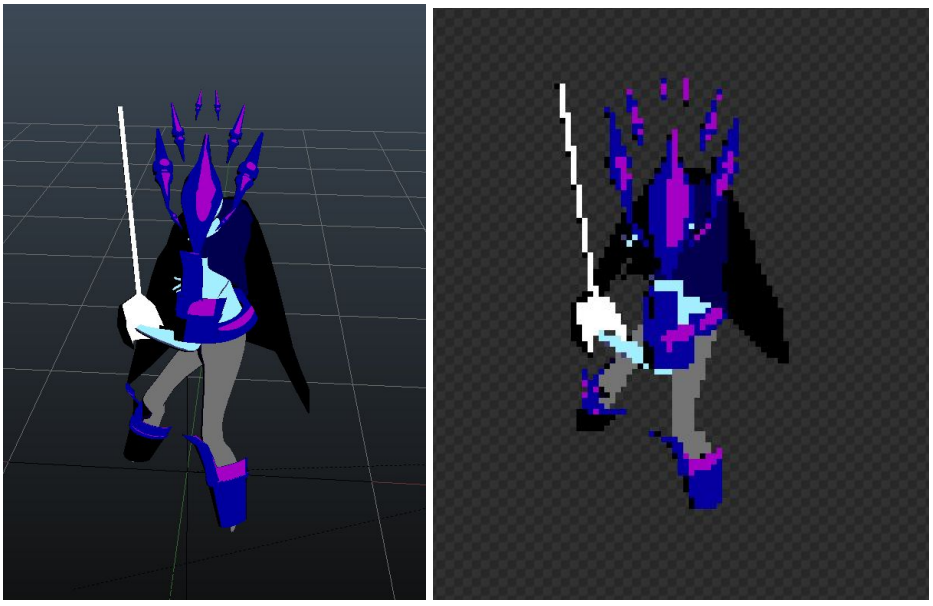
Otro de los interrogantes a resolver fue el de la tasa de fotogramas por segundo (*fps*) que deberían poseer las animaciones. Al ser un juego veloz y de precisión, se comenzó a trabajar con animaciones a 60 *fps*. Sin embargo, pronto quedó claro que este no era el camino a seguir: una tasa de fotogramas tan alta (algo propio de gráficos punteros) no concordaba con la resolución mínima del pixel art, dando lugar a una sensación antinatural al ojo. Por este motivo, se bajó la tasa de *fps* a 30, un punto intermedio que permitiera la fluidez buscada en el juego, pero que no desentonara con el pixel-art.

5.2 Producción

5.2.1 Enemigos

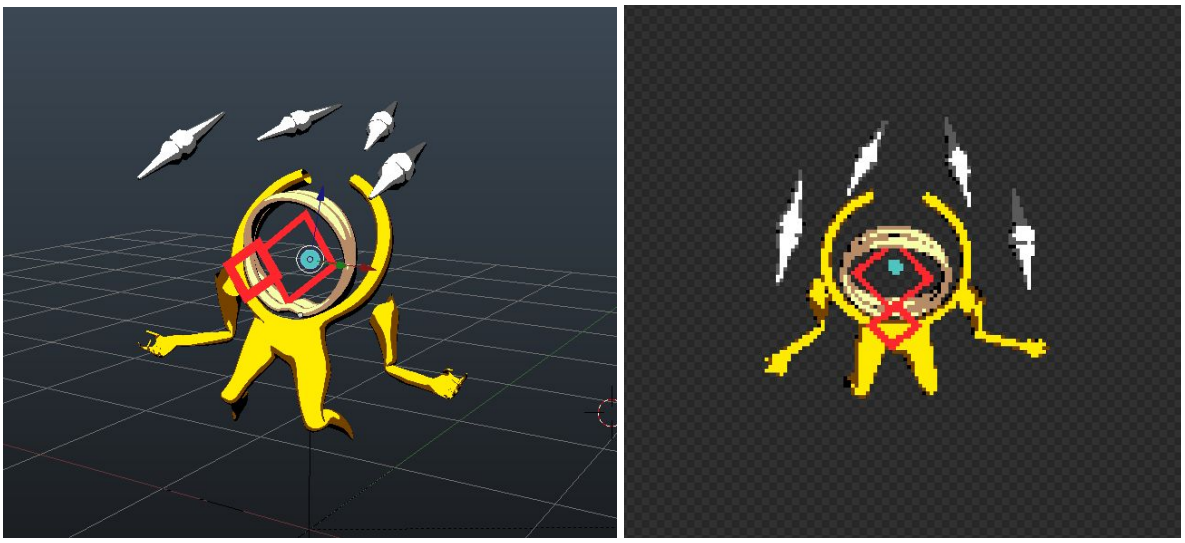
A lo largo de la producción, se crearon tres enemigos, adicionalmente a los dos generados durante preproducción. Todos ellos siguieron el patrón de creación establecido previamente.

Fantasma: esquiva todos los ataques cuerpo a cuerpo. Ataca teletransportandose al jugador. Pensado para incentivar la mecánica de contraataque.



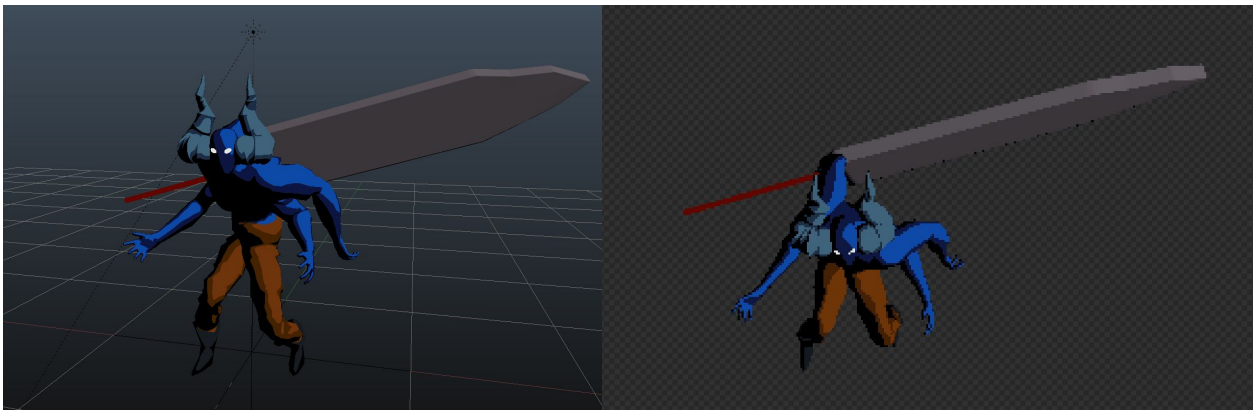
Fantasma (modelo y sprite)

Ojo: enemigo estático que dispara un láser en línea recta. Creado para forzar el movimiento del jugador de una manera distinta al Mago.



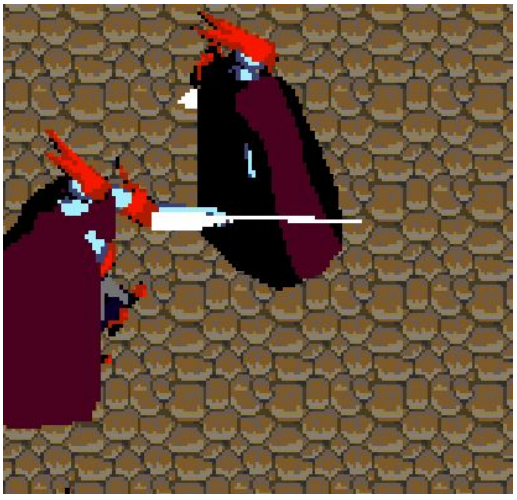
Ojo (modelo y sprite)

Demonio de cuatro brazos: el que sería el jefe final del juego, pensado como clímax. En lugar de un solo comportamiento definido, dispone de varios patrones de ataque, que lo hacen más desafiante que el resto.



Demonio de cuatro brazos(modelo y sprite)

Existe un cuarto enemigo, el **Cazador**, que fue descartado después de varios testeos.



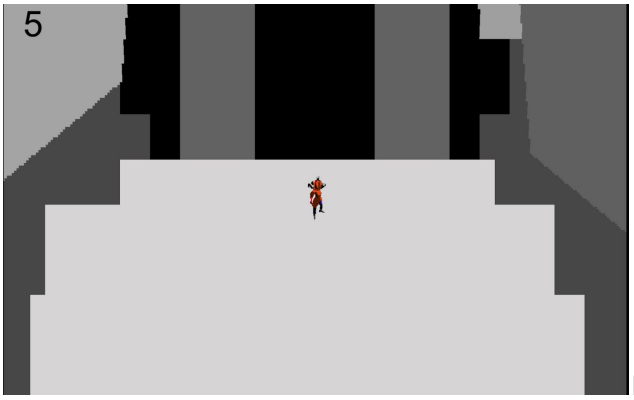
Cazador: enemigo descartado

Este enemigo también se teletransportaba si el jugador intentaba atacarlo. Su patrón ofensivo consistía en alinearse con el jugador y dispararle.
Sin embargo, este ataque solía fallar (al requerir que jugador y enemigo estuvieran alineados), y acabar con el Cazador resultaba engorroso (por tener una ventana de vulnerabilidad muy pequeña). Dado esto, no resultaba un enemigo ni interesante ni divertido.
El comportamiento del teletransporte se reaprovechó, con unos pocos cambios en el modelo, para crear al **Fantasma**. Asimismo, el patrón de ataque en línea recta fue el germen de lo que más tarde acabaría siendo el **Ojo.**

5.2.2 Escenarios

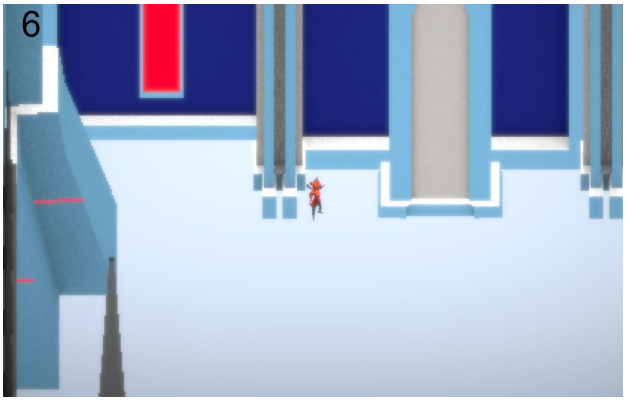
Inicialmente, se planeó separar los escenarios en dos grupos, uno de ambientación exterior y otro interior, con alrededor de tres niveles cada uno (mas la escena del jefe final).
Sin embargo, el poco tiempo disponible llevó a descartar la posibilidad de crear una segunda ambientación.

El proceso de crear escenarios ya ha sido explicado en este documento en el apartado de pre-producción. A continuación, se pueden apreciar las diferentes fases por las cuales pasó el primer nivel:

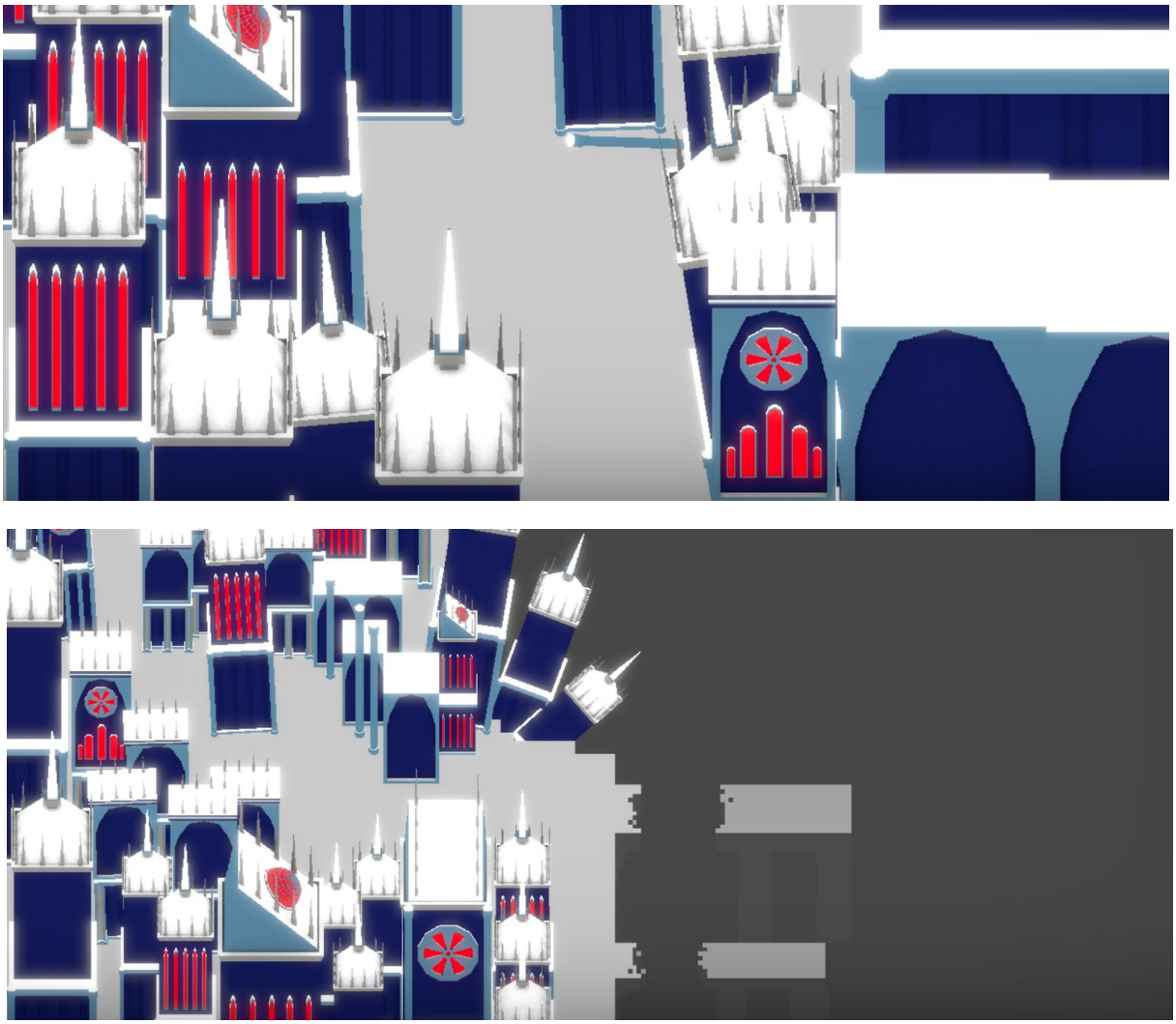


Primer nivel en la version 0.5

Primer nivel en la version 0.6



Primer nivel en la version 0.7



Diferentes capturas del escenario definitivo (niveles 1 y 2)

La paleta de colores ha sido elegida por contraste, para huir de los grises y marrones que normalmente pueblan las edificaciones góticas en las que el juego se basa. La tríada de colores de azul oscuro, blanco y rojo resulta vibrante y llamativa, sin dejar de provocar esa sensación de ominosidad que se buscaba en el art-decó urbano.

5.2.3 Audio

La mayoría de efectos de sonido fueron extraídos de páginas como Soundsnap y FreeAudio.
Ningún efecto fue retocado en profundidad, pero sí que se combinaron algunos (usando Audacity y Wwise) para crear riqueza.
Dentro de Wwise, muchos de los efectos (en especial, los que suenan con más frecuencia) se aleatorizaron, creando entre tres y cinco versiones del mismo sonido, con variaciones de *pitch* (balance de graves-agudos), para evitar la repetición.

La música (en total, dos pistas de audio) fue compuesta (de manera gratuita) por Guillermo Otero, historiador y músico. Cada pieza consta de dos partes: una introducción que se reproduce al iniciar la pista, y un cuerpo que se reproduce en bucle de manera infinita.

El estilo musical mezcla estilos corales con tecno, usando como referencia a la banda Juno Reactor o a Don Davis (compositores, entre otras cosas, de la banda sonora de la saga cinematográfica *Matrix*). Previamente, se había intentado contactar con Juno Reactor para solicitar el uso de su material, pero no se recibió respuesta alguna.

5.2.4 Playtesting

Tal y como se mencionaba en el apartado 4.3: Métodos de validación, todas las versiones estables del proyecto se testearon.

Overdriver_v0.0.2	29/08/2018 16:28
Overdriver_v0.0.3	16/12/2018 21:45
Overdriver_v0.0.4	29/12/2018 13:32
Overdriver_v0.0.5	08/01/2019 19:22
Overdriver_v0.0.6	27/01/2019 13:08
Overdriver_v0.0.7	06/02/2019 12:34
Overdriver_v0.0.8	10/02/2019 16:13
Overdriver_v0.0.9	13/02/2019 18:46
Overdriver_v0.1.0	17/02/2019 22:50
Overdriver_v0.1.1	21/02/2019 23:26
Overdriver_v0.2.0	02/03/2019 13:21
Overdriver_v0.2.1	03/03/2019 2:39
Overdriver_v0.3.0	07/03/2019 17:19
Overdriver_v0.3.5	10/03/2019 18:46
Overdriver_v0.4	14/03/2019 22:42
Overdriver_v0.4.1	21/03/2019 13:43
Overdriver_v0.5	02/04/2019 11:22
Overdriver_v0.5.5	06/04/2019 16:22
Overdriver_v0.5.6	10/04/2019 19:23
Overdriver_v0.6	20/04/2019 16:52
Overdriver_v0.6.1	27/04/2019 18:22

Las diferentes versiones de Demon Overdrive

Sin embargo, existe una desviación con respecto a la planificación original: los testeos externos programados son siguieron el patrón escrito. En lugar de testear externamente solo ciertas versiones (preproducción, alpha, beta y gold), se testeaban todas las estables, salvo algunas excepciones.
Esto se debe a dos motivos: primero, que la información recogida en estos testeos resultaba demasiado valiosa para el desarrollo; y segundo, resultaba complicado reunir a un gran número de personas para un día específico, dado que debía contar con personal ajeno al proyecto para realizar los tests.

Los resultados de los playtestings (y los cambios que propiciaron) fueron, sin seguir un orden cronológico:

- Buena sensación de combate. Posteriores cambios a la velocidad y el ritmo fueron revocados, ya que se prefería el inicial.
- Ajuste de la velocidad de esquive, depués de que muchos testers expresaran que les parecía excesiva.
- Ligado al punto anterior, muchos jugadores ignoraban a los enemigos y atravesaban el nivel esquivando. Debido a esto, se crearon “muros” que bloqueaban ciertas salas hasta que todos los enemigos en ella eran eliminados.
- Pequeños cambios estéticos en el escenario, en especial la eliminación de elementos de color rojo que no causaban daño (ya que tendían a confundir a los jugadores).
- Recolocación y balanceo del número de enemigos, para ajustar la curva de dificultad.
- Reporte de numerosos bugs, en especial de físicas.

5.3 Post-producción

La fase de post-producción, que comprende desde la versión 0.7 (Alpha) del proyecto hasta su finalización, transcurrió sin remarcable novedad.
No se crearon mecánicas nuevas ni añadió contenido per-se (salvo con la excepción de elementos de interfaz y ciertos menús).

Las principales tareas que se llevaron a cabo fueron:

- Modelo de la protagonista actualizado
- Arte de los escenarios finalizado
- Adición de efectos (como partículas y luces)
- Adición de animaciones faltantes (en especial, las de muerte)
- Adición de menú principal, pantalla de *Game Over* y pantalla de resultados
- Audio definitivo, incluyendo las pistas de música ambiental (compuestas por Guillermo Otero)
- Diversos pequeños detalles, como sombreado dinámico en los edificios (para enaltecer la sensación de profudidad) o rastro en los ataques a distancia
- *Bugfixing* intenso

Es recomendable remarcar que, aunque no hubo ninguna creación significativa de contenido, la apreciación de calidad del juego aumentó considerablemente (sobre todo, a nivel gráfico y estético).

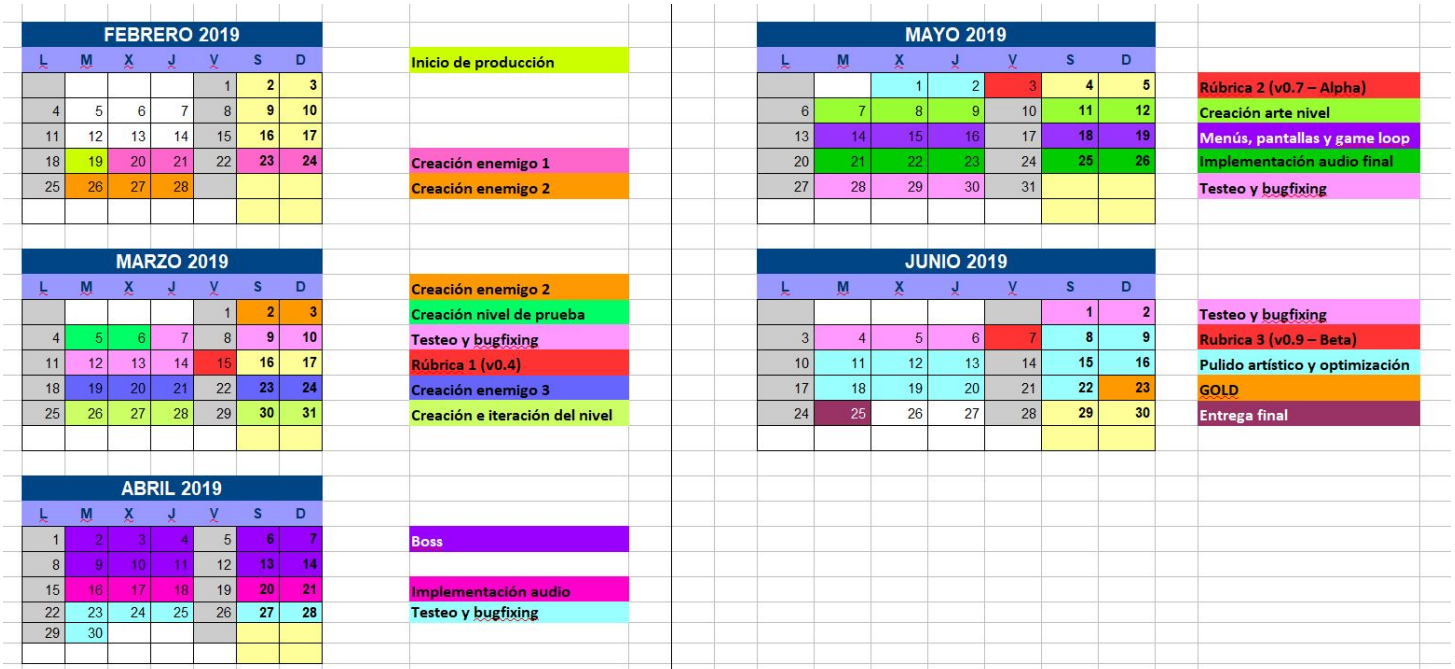
5.4 Cambios en la metodología

El desarrollo del proyecto no sufrió cambios en su metodología fundamental. Todos los procesos y herramientas empleadas resultaron inalterados.

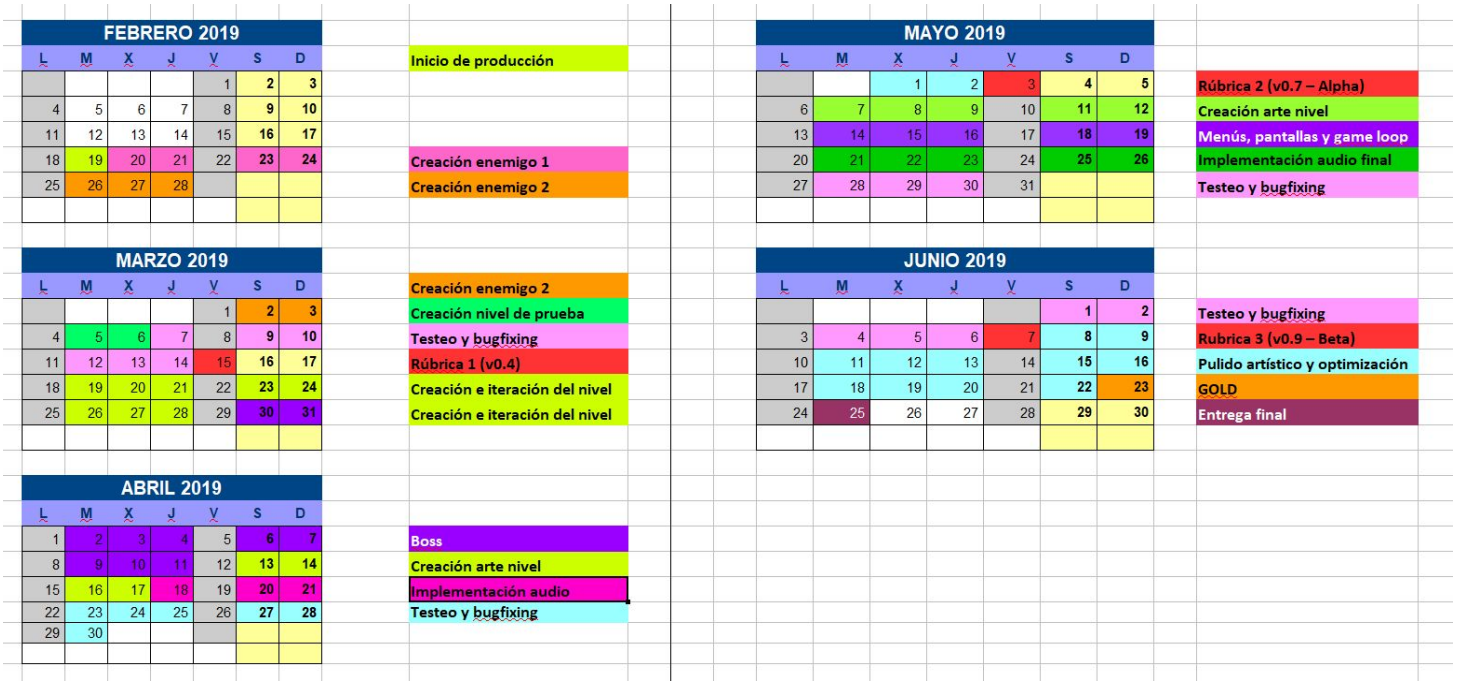
5.5 Desviaciones de la planificación

Si existieron, sin embargo, algunos cambios en el calendario.

Calendario inicial:



Como se distribuyó el tiempo realmente:



FEBRERO 2019							
L	M	X	J	V	S	D	
				1	2	3	Inicio de producción
4	5	6	7	8	9	10	
11	12	13	14	15	16	17	
18	19	20	21	22	23	24	Creación enemigo 1
25	26	27	28				Creación enemigo 2

MARZO 2019							
L	M	X	J	V	S	D	
				1	2	3	Creación enemigo 2
4	5	6	7	8	9	10	Creación nivel de prueba
11	12	13	14	15	16	17	Testeo y bugfixing
18	19	20	21	22	23	24	Rúbrica 1 (v0.4)
25	26	27	28	29	30	31	Creación e iteración del nivel
							Creación e iteración del nivel

ABRIL 2019							
L	M	X	J	V	S	D	
1	2	3	4	5	6	7	Boss
8	9	10	11	12	13	14	Creación arte nivel
15	16	17	18	19	20	21	Implementación audio
22	23	24	25	26	27	28	Testeo y bugfixing
29	30						

MAYO 2019							
L	M	X	J	V	S	D	
		1	2	3	4	5	Rúbrica 2 (v0.7 – Alpha)
6	7	8	9	10	11	12	Creación arte nivel
13	14	15	16	17	18	19	Menús, pantallas y game loop
20	21	22	23	24	25	26	Implementación audio final
27	28	29	30	31			Testeo y bugfixing

JUNIO 2019							
L	M	X	J	V	S	D	
					1	2	Testeo y bugfixing
3	4	5	6	7	8	9	Rubrica 3 (v0.9 – Beta)
10	11	12	13	14	15	16	Pulido artístico y optimización
17	18	19	20	21	22	23	GOLD
24	25	26	27	28	29	30	Entrega final

Los cambios mas importantes fueron:

- No se llegó a crear el quinto tipo de enemigo. Esta decisión fue tomada rápidamente, tras evaluar que se había reservado poco tiempo para la creación de los niveles, respondiendo a la filosofía de “calidad antes que cantidad”.
- Asimismo, se aumentó el tiempo dedicado al arte y al diseño de los niveles, ya que se consideró (acertadamente) que dos semanas no serían suficientes para generar niveles de calidad.

Es necesario hacer notar que, aunque se planificaban las semanas para una sola tarea, resultaba común que días concretos se dedicaran a tareas de otra índole (por ejemplo, durante la primera semana reservada a audio, se realizaron tareas de optimización).
Normalmente, esto respondía a que la tarea planificada se había terminado antes de tiempo; o bien, que existía algún algún problema que condicionaba al resto del desarrollo, y que exigía acción prioritaria.

En retrospectiva, a pesar de los ligeros cambios en la planificación, el calendario se respetó en su mayoría.

6. Conclusiones y futuros proyectos

6.1 Análisis de la técnica empleada

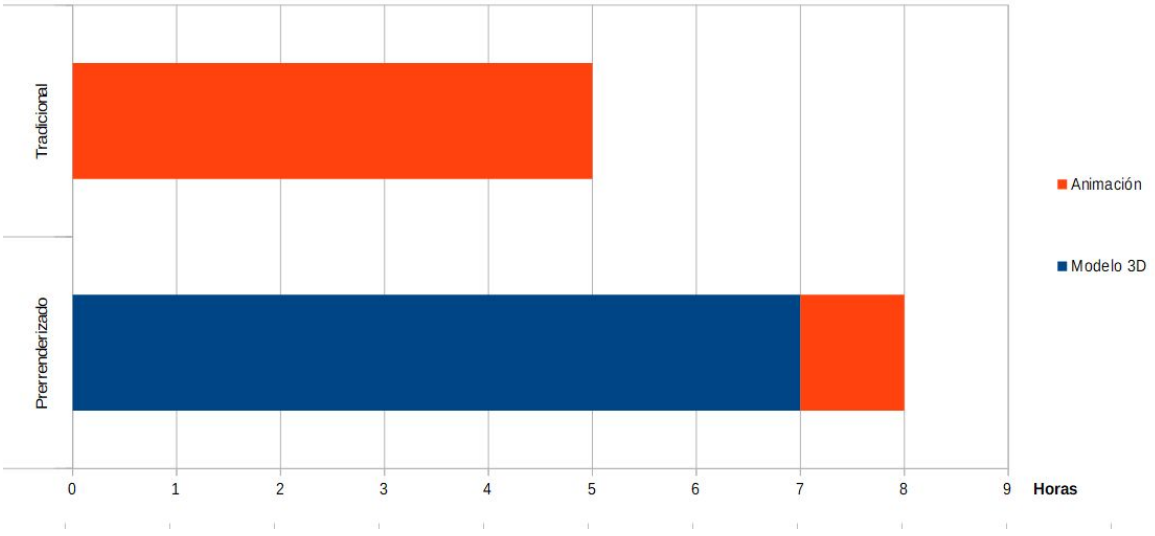
La técnica de prerrenderizado a baja resolución no solo ha alcanzado las expectativas esperadas: las ha superado con creces. La velocidad incrementada del proceso (sobre todo, a la hora de iterar elementos ya creados) ahorró no ya horas de trabajo, sino días enteros.

Si bien es cierto que esta técnica requiere de una inversión de tiempo inicial a la hora de crear el modelo 3D, se compensa claramente en etapas posteriores. gracias a la generación automática de fotogramas, la posibilidad de reusar objetos y modelos, y la facilidad para cambiar colores y propiedades (que la mayoría de las veces consiste en pulsar un botón)

Por poner un ejemplo práctico con una de las animaciones de la protagonista.

Asumamos que hemos partido de cero, y hemos tenido que crear el modelo. Estimemos que hemos tardado, aproximadamente, 7 horas en crear al personaje completo (es un modelo sencillo, con materiales simples y sin texturas).

Queremos tener una animación de 30 fotogramas por segundo, a una resolución 256 x 256 píxeles (relativamente elevada para un estilo pixel-art). La animación dura 2 segundos: en total, 60 fotogramas.
Estimando (de manera muy generosa) unos 5 minutos por fotograma, tardaríamos 5 horas en obtener esos 2 segundos de animación.

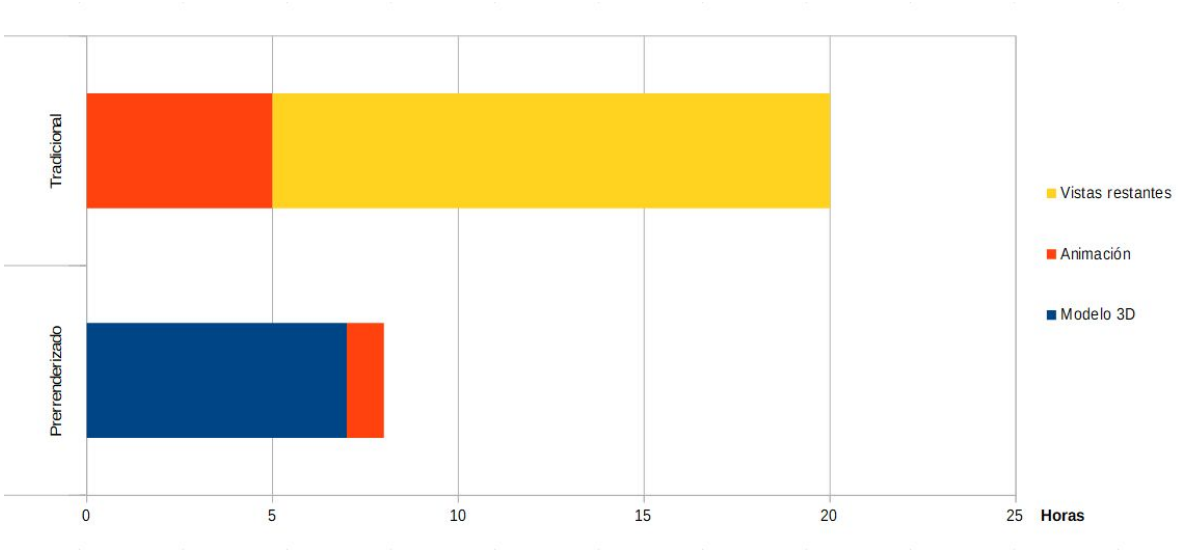


Comparación tradicional vs prerrenderizado (1)

Con la técnica de prerrenderizado, solo tendríamos que colocar el modelo en las posiciones clave, y el programa se encargaría de interpolar entre ellas, tras lo cual modificaríamos velocidades y añadiríamos detalles. Vamos a estimar que tardamos una hora en atravesar todo el proceso. Como los fotogramas que no son claves se generan automáticamente interpolando los que si lo son, la cantidad de fotogramas a generar nos es irrelevante.

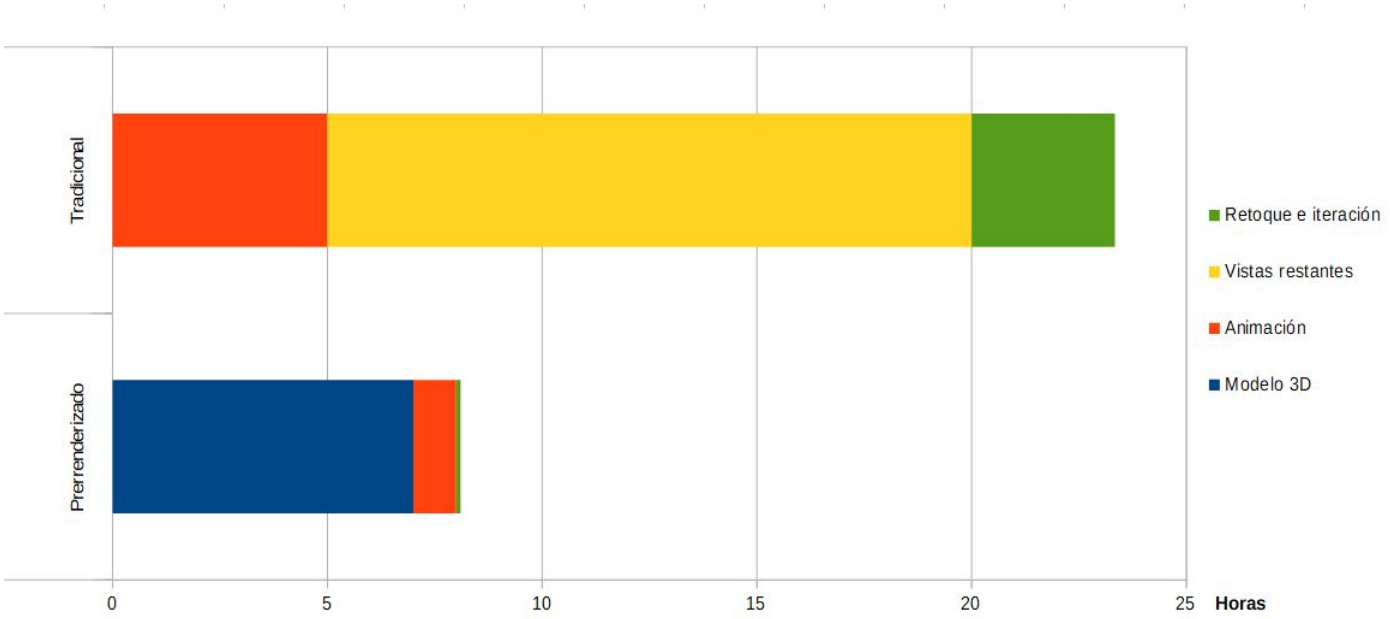
Ahora bien: en el caso de *Demon Overdrive*, necesitamos cuatro versiones de la misma animación, ya que se trata de un juego en vista cenital.

Por el método tradicional, siguiendo con la animación previa, alcanzaríamos las 20 horas de trabajo.
Por el método del prerrenderizado, el coste temporal seguiría siendo el mismo, ya que el código que se usa para procesar los fotogramas se encarga de hacerlo desde cuatro vistas diferentes del modelo.
El ahorro se hace patente.



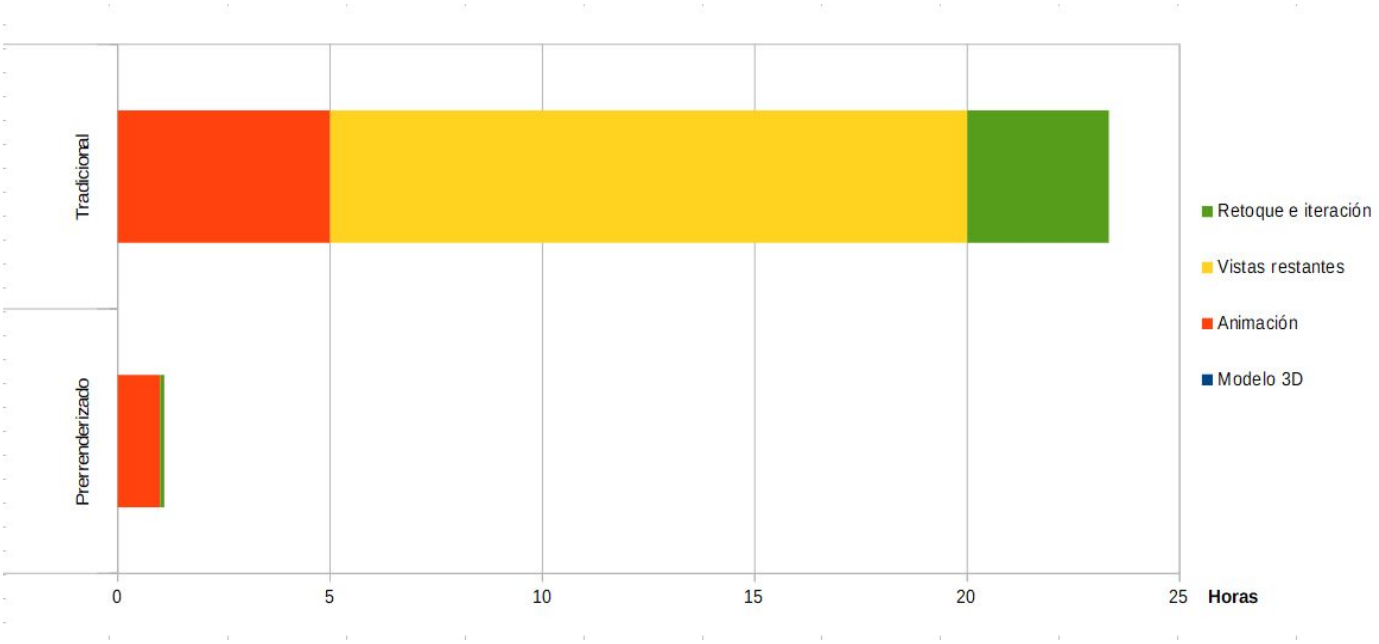
Comparación tradicional vs prerrenderizado (2)

Y, a mayores, ¿que pasa si queremos modificar una animación, por ejemplo, si resulta muy corta y queremos añadirle fotogramas?
Por el método tradicional, tendríamos que crearlos todos a mano, incluyendo los de las tres variantes adicionales para las distintas direcciones. Si quisieramos añadirle, digamos, diez fotogramas (un tercio de segundo), haciendo las mismas estimaciones que arriba, tardaríamos ***cinco minutos por fotograma x diez fotogramas x cuatro direcciones = 200 minutos,*** lease, algo más de tres horas.
Por el método del prerrenderizado, es tan sencillo como mover el fotograma clave a su nueva posición y volver a pulsar el botón de renderizado. El coste depende de los cambios que se quieran aplicar a la animación, pero si tan solo es una modificación de longitud, el coste es, a lo sumo, cinco minutos.



Comparación tradicional vs prerrenderizado (3)

Todo esto, sin tener en cuenta el ahorro que supone que, en posteriores animaciones, se eliminaría el coste temporal de crear de nuevo al personaje.



Comparación tradicional vs prerrenderizado (4)

Además, conseguir resultados en tan poco tiempo permite al artista invertir mucho más en retocar, pulir y mejorar cada animación. No solo se consiguen animaciones más rápido, sino de mayor calidad.

Teniendo en cuenta la gran cantidad de animaciones que se crearon para este proyecto, no es descabellado afirmar que el uso del prerrenderizado supuso un ahorro de meses de trabajo.

6.2 Post-mortem

6.2.1 ¿Que salió bien?

- El proyecto, a nivel general, se puede considerar un éxito. Queda demostrado que se pueden crear juegos de diferentes géneros y perspectivas de esta manera. Este proyecto no hubiera sido posible si no se hubiera empleado esta técnica.
- El juego alcanzó la calidad deseada. El *gameplay* rápido y sencillo, unido a un estilo estético llamativo, causó excelentes sensaciones entre los jugadores y comentarios muy positivos.
- Las herramientas utilizadas resultaron (salvo pequeños detalles) apropiadas: el flujo de trabajo no se vió alterado prácticamente en ningún momento por causas técnicas, y ningún resultado se vio condicionado por los programas. Unity resultó ser un excelente motor, intuitivo y flexible.

6.2.2 ¿Que salió mal?

- La fluidez de las animaciones (30 fotogramas por segundo) resultaba extraña, estéticamente hablando, para muchos de los jugadores. Muchos de ellos hicieron notar que, efectivamente, aún se notaba que eran modelos 3D. Dada la naturaleza veloz del juego, esto no supuso ningún problema. Sin embargo, esto hace ver que la técnica aún puede refinarse. Una de las posibles opciones sería la de reducir la tasa de fotogramas a 12 por segundo.
- Debido a ser un equipo de una sola persona, no se llegó a dedicar tanto tiempo como el deseado al diseño de niveles.
- Por este mismo motivo, el apartado de arte acabó por debajo de las expectativas (aunque la paleta de colores causó buenas sensaciones, y los efectos de post-procesado mejoraron la experiencia notablemente) .
- A pesar de que el juego resultara satisfactorio y divertido para los que lo probaron, no se llegó a lograr un uso variado de todas las mecánicas disponibles. Todos los jugadores se adherían a una de ellas, y la explotaban hasta la saciedad. Por suerte, estas mecánicas variaban de jugador a jugador, así que no se debería considerar un completo fracaso en este aspecto.
- El apartado sonoro tampoco llegó a la calidad deseada. No por falta de tiempo, sino por falta de material bruto de calidad: muchas de las pistas de

audio disponibles procedían de diferentes fuentes, con diferentes compresiones, lo que resultó en un sonido de calidad irregular.

6.3 Futuros proyectos

Dado que la técnica del prerrenderizado en baja resolución ha resultado un éxito, ahorrando cantidades espectaculares de tiempo, no solo la usaría en futuros proyectos, sino que, probablemente, no vuelva a crear pixel-art de manera tradicional.
El único cambio que aplicaría al proceso consistiría, como se ha explicado en el post-mortem, rebajar la tasa de fotogramas por segundo (posiblemente a 12).

El mayor cambio que experimentarían futuros proyectos sería la composición del equipo. A pesar de que el prerrenderizado ha acelerado el proceso enormemente, crear todo el apartado artístico de un juego sigue siendo demasiado volumen de trabajo para que lo lleve a cabo la misma persona que programa y diseña.

A pesar de todo, soy optimista al pensar en la gran cantidad de contenido (de mucha mayor calidad que el que yo he creado en este proyecto) que un artista dedicado podría crear en tiempo récord empleando esta técnica.

Por ello mismo, es poco probable que vuelva a desarrollar un juego yo solo.

7. Bibliografia

-*Gamasutra: Using a 3D pipeline to create de art of Dead Cells*, consultado el 25 de Agosto de 2018

-*Entrevista a Motion Twin*, consultado el 5 de Marzo de 2019

-*Pixel Art Speedpaint*, consultado el 5 de Marzo de 2019

-*Unity API*, consultado repetidamente desde el 1 de Septiembre de 2018

-*Blender API*, consultado repetidamente desde el 1 de Septiembre de 2018

-*No Copyright Music*, consultado el 14 de Marzo de 2019

-*Página web de la banda Juno Reactor*, consultada el 12 de Febrero de 2019

-*Página web del compositor Don Davis*, consultada el 12 de Febrero de 2019

-*Game programming Patterns*, libro consultado repetidamente desde el inicio hasta el final del proyecto